ФЕДЕРАЛЬНОЕ АГЕНТСТВО

ПО ТЕХНИЧЕСКОМУ РЕГУЛИРОВАНИЮ И МЕТРОЛОГИИ

НАЦИОНАЛЬНЫЙ СТАНДАРТ РОССИЙСКОЙ ФЕДЕРАЦИИ

ГОСТ Р МЭК 62628— 2021

Надежность в технике

РУКОВОДСТВО ПО ОБЕСПЕЧЕНИЮ НАДЕЖНОСТИ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

(IEC 62628:2012, Guidance on software aspects of dependability, IDT)

Издание официальное

Москва
Российский институт стандартизации
2021

Предисловие

1 ПОДГОТОВЛЕН Закрытым акционерным обществом «Научно-исследовательский центр контроля и диагностики технических систем» (ЗАО «НИЦ КД») на основе собственного перевода на русский язык англоязычной версии стандарта, указанного в пункте 4

2 ВНЕСЕН Техническим комитетом по стандартизации ТК 119 «Надежность в технике»

3 УТВЕРЖДЕН И ВВЕДЕН В ДЕЙСТВИЕ Приказом Федерального агентства по техническому регулированию и метрологии от 21 сентября 2021 г. № 990-ст

4 Настоящий стандарт идентичен международному стандарту МЭК 62628:2012 «Руководство по обеспечению надежности программного обеспечения» (IEC 62628:2012 «Guidance on software aspects of dependability», IDT).

Международный стандарт разработан Техническим комитетом IEC/TC 56.

Наименование настоящего стандарта изменено относительно наименования указанного международного документа для приведения в соответствие с ГОСТ Р 1.5—2012 (пункт 3.5).

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.rst.gov.ru)

Содержание

Введение

В настоящее время программное обеспечение нашло широкое применение в современной продукции и системах. Примеры включают программное обеспечение в программируемых средствах управления, компьютерных системах и сетях связи. За прошедшие годы разработано много стандартов по созданию программного обеспечения, управлению программным процессом, обеспечению качества и безотказности программного обеспечения, но только в нескольких стандартах рассмотрены проблемы программного обеспечения с точки зрения надежности.

Надежность — это способность системы работать в соответствии с установленными требованиями и целями в заданных условиях использования. Надежность системы подразумевает, что система заслуживает доверия и способна предоставлять требуемые услуги по запросу для удовлетворения потребностей пользователя. Современные тенденции в области применения программных средств в сфере услуг привели к внедрению интернет-услуг и веб-разработок. Стандартизованные интерфейсы и протоколы позволяют использовать сторонние функции программного обеспечения третьей стороны через интернет, чтобы разрешить применение межплатформных, межпровайдерных и междоменных приложений. Программное обеспечение стало движущим механизмом функционирования сложных систем и обеспечения жизнеспособного электронного бизнеса для беспрепятственной интеграции и управления процессами организации. Разработка программного обеспечения играет основную роль в обработке данных, мониторинге безопасности, обеспечении охраны и защиты и коммуникативных связях услуг сети. Это изменение парадигмы привело к ситуации, когда мировое бизнес-сообщество в значительной степени применяет и зависит от систем программного обеспечения для поддержания работы бизнеса. Надежность программного обеспечения играет доминирующую роль в обеспечении успеха работы системы и целостности данных.

В настоящем стандарте приведены современные лучшие отраслевые практики и представлены соответствующие методы обеспечения надежности программного обеспечения. В стандарте определено влияние управления на аспекты надежности программного обеспечения и приведены соответствующие технические процессы разработки надежного программного обеспечения, используемого в системах. Эволюция программных технологий и быстрая адаптация применения программных средств в отраслевой практике сформировали потребность в практическом стандарте по надежности программного обеспечения для глобальной бизнес-среды. В настоящем стандарте представлен структурный подход к применению методов обеспечения надежности программного обеспечения.

В стандарте представлены общие требования и процессы обеспечения надежности программного обеспечения. Они формируют основу применения надежности к большинству программных продуктов и созданию систем программного обеспечения. Дополнительные требования необходимы для применения программных средств при выполнении критически важных задач и обеспечении безопасности и охраны. Вопросы соответствия отраслевых программных средств требованиям безотказности и качества в настоящем стандарте не рассмотрены.

Стандарт может служить руководством по проектированию надежности встроенного программного обеспечения. Однако в нем не рассмотрены аспекты реализации встроенного программного обеспечения с программным обеспечением, содержащимся в микросхемах, для реализации специализированных функций. Примеры включают интегральные схемы специального назначения (ASIC) и управляющие устройства с микропроцессорным управлением. Такие программные продукты часто разрабатывают и интегрируют как часть оборудования для того, чтобы минимизировать его размер, вес и обеспечить работу в реальном времени, такую как в сотовых телефонах. Хотя общие принципы и методы обеспечения надежности, описанные в настоящем стандарте, могут быть использованы при проектировании и применении встроенного программного обеспечения, необходимы особые требования к их физическим носителям, изготовлению устройств и внедрению встроенного программного обеспечения. Физика отказов аппаратных средств отличается от отказов систем программного обеспечения.

Настоящий стандарт не предназначен для оценки соответствия или сертификации.

НАЦИОНАЛЬНЫЙ СТАНДАРТ РОССИЙСКОЙ ФЕДЕРАЦИИ

Надежность в технике

РУКОВОДСТВО ПО ОБЕСПЕЧЕНИЮ НАДЕЖНОСТИ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

Dependability in technics. Guidance on provision of software dependability

Дата введения — 2022—01—01

1 Область применения

В настоящем стандарте рассмотрены вопросы надежности программного обеспечения и приведены рекомендации по обеспечению надежности программного обеспечения с помощью управления, процессов проектирования и условий применения. В стандарте установлены общая структура требований к надежности программного обеспечения, процесс обеспечения надежности программного обеспечения на всех этапах жизненного цикла системы, критерии и методы обеспечения надежности при разработке и изготовлении программного обеспечения и представлены практические подходы для оценки показателей надежности программного обеспечения и систем, включающих программное обеспечение.

Настоящий стандарт предназначен для разработчиков и поставщиков систем программного обеспечения, интеграторов систем, операторов и специалистов по сопровождению, а также пользователей систем программного обеспечения, заинтересованных в разработке надежных программных средств и систем.

2 Нормативные ссылки

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

IEC 60050-191, International Electrotechnical Vocabulary — Chapter 191: Dependability and quality of service (Международный электротехнический словарь. Глава 191. Надежность и качество услуг)

IEC 60300-3-15, Dependability management — Part 3-15: Application guide — Engineering of system dependability (Менеджмент надежности. Часть 3-15. Прикладное руководство. Разработка надежности систем)

3 Термины, определения и сокращения

В настоящем стандарте применены термины по МЭК 60050-191, а также следующие термины с соответствующими определениями:

3.1 Термины и определения

3.1.1 **программное обеспечение** (software): Программы, процедуры, правила, документация и данные системы обработки информации.

Примечание 1 — Программное обеспечение — это интеллектуальный продукт, который не зависит от носителя.

Примечание 2 — Для работы программного обеспечения, выполнения программ, хранения и передачи данных необходимы аппаратные средства.

Издание официальное

Примечание 3 — Типы программного обеспечения охватывают прошивку, системное программное обеспечение и прикладное программное обеспечение.

Примечание 4 — Документация программного обеспечения включает в себя: спецификации требований, спецификации проекта, распечатку исходных кодов, комментарии к исходным кодам, тексты «справок» и сообщений для отображения на интерфейсе «человек/компьютер», инструкции по установке, инструкции по эксплуатации, руководство пользователя и руководство по сопровождению, используемое при обслуживании программного обеспечения.

3.1.2 **прошивка** (firmware): Программное обеспечение, содержащееся в постоянном запоминающем устройстве и не предназначенное для модификации.

ПРИМЕР — Базовая система ввода/вывода (BIOS) персонального компьютера.

Примечание — Модификация такого программного обеспечения требует замены или перепрограммирования содержащего его аппаратного устройства.

3.1.3 **встроенное программное обеспечение** (embedded software): Программное обеспечение, применяемое в системе, основная цель которой не является вычислительной.

ПРИМЕР — Программное обеспечение, используемое в системе управления двигателем или в системах управления тормозами транспортных средств.

3.1.4 **программный блок; программный модуль** (software unit, software module): Элемент программного обеспечения, который может быть отдельно скомпилирован в программных кодах для выполнения задачи или действия по достижению желаемого результата функции или функций программного обеспечения.

Примечание 1 — Термины «модуль» и «блок» часто используют как синонимы или как субэлементы друг друга (по-разному) в зависимости от контекста. Взаимосвязь этих терминов еще недостаточно стандартизирована.

Примечание 2 — В идеальной ситуации программный блок может быть спроектирован и запрограммирован для выполнения только определенной функции. В некоторых приложениях может потребоваться два или более блоков программного обеспечения, объединенных для достижения конкретной функции программного обеспечения. В таких случаях блоки программного обеспечения тестируют как единую функцию программного обеспечения.

3.1.5 **объект конфигурации программного обеспечения** (software configuration item): Объект программного обеспечения, скомпонованный и рассматриваемый как единый объект в процессе управления конфигурацией.

Примечание — Объект конфигурации программного обеспечения может состоять из одного или нескольких блоков программного обеспечения, предназначенных для выполнения функции программного обеспечения.

3.1.6 **функция программного обеспечения** (software function): Элементарная операция, выполняемая модулем или блоком программного обеспечения в соответствии с установленными требованиями.

3.1.7 **система программного обеспечения** (software system): Определенный набор объектов программного обеспечения, которые интегрированы и совместно работают в соответствии с установленными требованиями.

ПРИМЕР — Прикладное программное обеспечение (программное обеспечение для учета и управления информацией); программное обеспечение для программирования (программное обеспечение для анализа работы и методов CASE), системное программное обеспечение (программное обеспечение для контроля и управления компьютерной системой, такой как операционные системы).

3.1.8 **надежность программного обеспечения** (software dependability): Способность программного блока работать в составе системы в соответствии с установленными требованиями.

3.1.9 **неисправность программного обеспечения, ошибка** (software fault, bug): Состояние объекта программного обеспечения, препятствующее его работе в соответствии с установленными требованиями.

Примечание 1 — Неисправности программного обеспечения — это ошибки спецификации, проектирования, программирования, встроенного компилятора или неисправности, возникшие при обслуживании программного обеспечения.

Примечание 2 — Неисправности программного обеспечения неактивны до тех пор, пока не активируются определенным триггером, и обычно переходят в неактивное состояние при отключении триггера.

Примечание 3 — В настоящем стандарте ошибка — это конкретный случай неисправности программного обеспечения, ошибку также называют скрытой неисправностью программного обеспечения.

3.1.10 **отказ программного обеспечения** (software failure): Отказ, представляющий собой проявление неисправности программного обеспечения.

Примечание — Единичная неисправность программного обеспечения продолжает проявляться как отказ до тех пор, пока неисправность не будет устранена.

3.1.11 **код** (code): Символьная или битовая комбинация, которой присваивают определенное значение для представления компьютерной программы на языке программирования.

Примечание 1 — Исходные коды представляют собой кодированные инструкции и определения данных, выраженные в форме, подходящей для ввода в ассемблер, компилятор или другой транслятор.

Примечание 2 — Кодирование — это процесс преобразования логики и данных, установленных в спецификации или описании проекта, на язык программирования.

Примечание 3 — Язык программирования — это язык, используемый для записи компьютерных программ.

3.1.12 **(компьютерная) программа** ((computer) program): Набор закодированных инструкций, предназначенных для выполнения установленных логических и математических операций с данными.

Примечание 1 — Программирование — это общая деятельность по разработке программного обеспечения, в рамках которой программист или пользователь компьютера устанавливает определенный набор инструкций, которые должен выполнять компьютер.

Примечание 2 — Программа состоит из комбинации закодированных инструкций и определений данных, которые позволяют компьютерному оборудованию выполнять вычислительные или контрольные функции.

3.2 Сокращения

В настоящем стандарте применены следующие сокращения:

ASIC — интегральная схема для конкретного применения;

CASE — автоматизированная разработка программного обеспечения;

CMM — модель зрелости возможностей;

CMMI — комплексная модель зрелости;

COTS — приобретаемая продукция;

FMEA — анализ видов и последствий отказов;

FTA — анализ дерева неисправностей;

IP — интернет-протокол;

IT — информационные технологии;

KSLOC — тысяча строк исходного кода;

ODC — классификация ортогональных дефектов;

RBD — структурная схема надежности;

USB — универсальная последовательная шина.

4 Анализ аспектов надежности программного обеспечения

4.1 Программное обеспечение и системы программного обеспечения

Программное обеспечение является виртуальным объектом. В настоящем стандарте программное обеспечение связано с процедурами, программами, кодами, данными и инструкциями по управлению системой и обработке информации. Система программного обеспечения состоит из интегрированных блоков программного обеспечения, таких как компьютерные программы, процедуры и исполняемые

коды, встроенных в аппаратное обеспечение по обработке и управлению для реализации работы и выполнения функции системы. Структура системы программного обеспечения может быть рассмотрена как структура, представляющая структуру системы и состоящая из программного обеспечения подсистем и блоков более низкого уровня. Программный блок может быть протестирован, как указано при разработке программы. В некоторых случаях для создания программной функции необходимо два или более блоков программного обеспечения. Система включает в себя элементы как аппаратного обеспечения, так и программного обеспечения, взаимодействующие между собой для обеспечения полезных функций при выполнении требуемых услуг.

В комбинированной аппаратно-программной системе блоки программного обеспечения системы выполняют две основные функции: a) операционное программное обеспечение для обеспечения непрерывной работы аппаратных элементов системы; b) прикладное программное обеспечение, которое запускают, как и когда это требуется по запросам пользователя для предоставления потребителю определенных услуг. Анализ надежности подсистем программного обеспечения должен учитывать факторы времени применения программного обеспечения в профиле эксплуатации системы и те элементы программного обеспечения, которые необходимы для работы системы в течение всего времени ее работы. Моделирование программного обеспечения необходимо для распределения вероятности безотказной работы по компонентам системы и оценки надежности систем, основанных на применении программного обеспечения.

Аспекты надежности человеческого фактора [1] играют ключевую роль в результативном проектировании и разработке программного обеспечения. Интерфейс человек-машина и операционная среда влияют на результат взаимодействия программного и аппаратного обеспечения и влияют на надежность работы системы. Это приводит к стратегической потребности в разработке надежного программного обеспечения и приложению усилий по сопровождению программного обеспечения в процессе жизненного цикла [2].

4.2 Надежность и организация работы программного обеспечения

Надежность программного обеспечения может быть достигнута путем правильного проектирования и соответствующей эксплуатации системы. В настоящем стандарте представлен подход, при котором существующие методы обеспечения надежности и лучшие отраслевые практики могут быть идентифицированы и использованы для проектирования и разработки надежного программного обеспечения. Системы менеджмента надежности [3, 4] описывают ситуации, когда соответствующие мероприятия по обеспечению надежности могут быть выполнены в процессе жизненного цикла. На достижение надежности программного обеспечения влияют:

- политика менеджмента надежности и техническое управление надежностью;
- процессы проектирования и разработки;
- конкретные требования проекта и условий применения.

Организации, занимающиеся разработкой программного обеспечения, представляют собой организованные и управляемые группы, имеющие технические средства и персонал. При этом установлены соответствующие обязанности, полномочия и взаимосвязи, включая программное обеспечение, как часть их обычной деятельности. Такие организации существуют в правительствах, общественных и частных объединениях, компаниях, ассоциациях и учреждениях. Такие организации структурированы в соответствии с конкретными потребностями бизнеса и условиями применения для различных комбинаций разработки, эксплуатации и предоставления услуг.

Обычно такие организации:

a) разрабатывают программное обеспечение как основную продукцию;
b) разрабатывают аппаратное обеспечение со встроенным программным обеспечением;
c) осуществляют сопровождение программных средств у потребителей;
d) эксплуатируют и обслуживают системы и сети программного обеспечения.

В приложении А приведено описание классификации программного обеспечения и его применений, обычно предоставляемых организациями, занимающимися разработкой, внедрением и поддержкой программного обеспечения.

4.3 Связь надежности программного обеспечения с надежностью аппаратных средств

С точки зрения надежности режим и характеристики работы программного обеспечения в оборудовании могут отличаться от режима и характеристик при испытаниях. Коды программного обеспече-

ния создают люди. Они могут включать ошибки человека, на которые влияют условия проектирования и культура организации. Не смотря на то, что большая часть данных об отказах компонентов аппаратных средств хорошо документирована и классифицирована в условиях использования, особенности неисправностей программного обеспечения и прослеживаемость их причин и последствий трудно определить при эксплуатации системы. В большинстве случаев неисправности программного обеспечения, приводящие к отказам системы, не могут быть последовательно воспроизведены. Корректирующие действия при отказах системы, вызванные неисправностями программного обеспечения, не гарантируют полного устранения основных причин проблем с программным обеспечением.

Ошибка после ее срабатывания приводит к отказу программного обеспечения (событию) и проявляется как неисправность программного обеспечения. Все неисправности программного обеспечения, которые вызывают неспособность программного обеспечения выполнять предназначенные функции, замечают пользователи программного обеспечения. Неисправности и ошибки приводят к тому, что программное обеспечение не работает в соответствии с назначением. Пользователь программного обеспечения, содержащего ошибки, которое еще может продолжать выполнять предназначенную функцию, может эти ошибки не заметить. Ошибки могут вызывать отказы, но также могут создавать проблемы, которые не влияют на определенную функцию. Неисправность программного обеспечения может вызвать отказ системы, который может проявляться в виде систематических признаков отказа.

Системы программного обеспечения и аппаратные объекты имеют много общего. Они проходят стадию проектирования и разработки, за которой следуют интеграция, испытания и производство (изготовление). Обнаружение отказов и скрытых неисправностей происходит в результате тщательного анализа, процессов тестирования и верификации с высоким уровнем охвата тестированием отказов. Высокий уровень охвата процессом проверки (тестирования) определяет оценка процента обнаружения неисправностей или вероятности обнаружения неисправностей. Хотя методы управления похожи, существуют и различия [5], [6]. Ниже приведены некоторые примеры:

- У программного обеспечения нет физических свойств, в отличие от аппаратных средств. Программное обеспечение не изнашивается. Отказы, связанные с неисправностью программного обеспечения, появляются без предварительных признаков и часто признаков их появления. У аппаратного обеспечения часто существует период постепенного износа и, возможно, очень медленного износа до достижения состояния отказа.

- Изменения в программном обеспечении являются гибкими и намного менее трудоемкими или дорогостоящими по сравнению с изменением конструкции оборудования. Изменения в конструкции оборудования требуют выполнения ряда трудоемких корректировок основного оборудования, закупки материалов, изготовления, сборки и документирования. Однако регрессионное тестирование больших и сложных программ может иметь временные ценовые ограничения.

- Проверка и испытания аппаратного обеспечения могут быть упрощены, поскольку при этом можно проводить ограниченные испытания благодаря знанию физики анализируемого устройства и прогнозированию его состояния. Тестирование программного обеспечения также может быть упрощено с помощью регрессионного тестирования и анализа для проверки незначительных изменений в программном обеспечении, вызванных выявленной причиной отказа. Тем не менее, незначительные изменения для исправления вероятных причин отказов программного обеспечения, таких как работа в условиях высокой нагрузки, могут привести к очень сложным циклам тестирования и проверки, для демонстрации адекватного исправления проблемы.

- Действия по ремонту и техническому обслуживанию позволяют восстановить работоспособность аппаратного обеспечения обычно без изменения конструкции. Восстановление и сопровождение программного обеспечения включает изменение структуры с новыми пакетами услуг и отключение программного обеспечения для исправления или устранения неисправностей программного обеспечения.

4.4 Взаимодействие программного обеспечения и аппаратных средств

Взаимодействие программного обеспечения и аппаратных средств происходит при работе системы. Существуют проблемы надежности в интерфейсе между оборудованием и операционной системой. Проблемы обычно решают путем включения методов обнаружения и исправления ошибок, обработки отключений оборудования и операционной системы для уменьшения физических неисправностей и ошибок информации и синхронизации, которые существуют при взаимодействии. Появление многоядерных процессоров позволяет использовать резервирование в системах для параллельной обработки данных и повышения надежности работы системы. Это позволяет пользователю, программисту

или разработчику системы влиять на нее и использовать резервирование, присущее многоядерным процессорам, для обнаружения ошибок и восстановления после них. Это также дает возможность восстановления после программных ошибок или кратковременных ошибок, которые влияют на аппаратное или программное обеспечение или на то и другое одновременно. При эксплуатации повышенной сложности многоядерных резервированных процессоров необходимо учитывать такие особенности.

Во всех системах управления система контролирует некоторые физические процессы реальных аппаратных устройств, таких как датчики и исполнительные механизмы, которые могут отказать при работе системы. Многие из этих устройств содержат встроенное программное обеспечение, недоступное для проектировщика и разработчика системы. Примером являются интеллектуальные датчики, которые включают обнаружение ошибки, резервирование и некоторые устройства исправления ошибок, которыми управляет встроенное программное обеспечение. Важно пересмотреть алгоритмы управления программным обеспечением. Необходимо обеспечить, чтобы алгоритмы управления были устойчивы к ошибочным показателям датчиков и пропущенным значениям сенсоров, и могли обнаружить отказ при активации и вернуться к устойчивому состоянию. Обратная связь с устройством необходима для подтверждения успешной активации. Механизм обратной связи должен содержать некоторую независимую проверку влияния заданного срабатывания. Особенности функционирования системы управления, предположения и виды отказов должны быть учтены при проектировании системы управления программным обеспечением.

Умышленное и злонамеренное введение неисправностей аппаратных средств может привести к нарушению в их работе или работе программных алгоритмов, если система подвергается преднамеренной кибератаке. Например, можно ввести аппаратные неисправности в криптографическую систему для извлечения ключа или внедрить вирус в USB-устройство машины для голосования. Взаимодействие программного и аппаратного обеспечения может создать серьезные проблемы в работе системы и повлиять на ее надежность

Проблемы совместимости, связанные с взаимодействием программного и аппаратного обеспечения, могут также возникать при неправильном повторном использовании программного обеспечения, использовании в других условиях или для другого применения.

Решение проблем надежности, связанных с взаимодействием программного и аппаратного обеспечения, состоит в улучшении понимания работы новой технологической системы и осторожности при выполнении оценки и испытаниях на надежность, чтобы полностью учесть влияние отказов оборудования на программную систему.

5 Надежность программного обеспечения: разработка и применение

5.1 Структура жизненного цикла системы

Организация должна установить структуру жизненного цикла системы, в соответствии с которой будет проведена разработка продукции и изготовление системы. Структуру используют для определения жизненного цикла системы и управления работой процессов жизненного цикла системы. В МЭК 60300-3-15 приведено описание обеспечения надежности системы и реализации жизненного цикла, основанное на технических процессах ИСО/МЭК 15288 [7]. Данную структуру применяют ко всем системам, состоящим из аппаратных средств, программных средств или и тех и других.

5.2 Обеспечение надежности программного обеспечения

Деятельность по разработке программного обеспечения на стадии проектирования и в течение срока его полезного использования необходимо планировать, координировать и управлять этой деятельностью вместе с соответствующими аппаратными объектами. Деятельность по проектированию в течение периода полезного использования может включать в себя изменения конструкции, вызванные высокой интенсивностью отказов при использовании потребителями или устареванием оборудования при обеспечении запасными частями для работы системы. Поскольку аппаратное обеспечение в течение жизненного цикла продукта изменяется, программное обеспечение также должно быть изменено. Это необходимо, так как проект системы требует прямой и обратной совместимости между различными версиями и конфигурациями проекта системы.

Деятельность по обеспечению надежности должна быть интегрирована в планы проекта и включена в задачи разработки системы для эффективного проектирования, изготовления, внедрения, эксплуатации и обслуживания системы. К настоящему стандарту также может быть применено руководство по

проектированию надежности систем, установленное в МЭК 60300-3-15. Руководство по обеспечению надежности программного обеспечения состоит из следующих рекомендуемых процедур:

a) определение целей применения программного обеспечения и требований, относящихся к жизненному циклу программного обеспечения (см. 5.3) и условиям его применения (см. А.2);

b) определение применимых показателей надежности программного обеспечения (см. 5.4), относящихся к проекту программного обеспечения;

c) анализ адекватности процессов управления надежностью и наличия ресурсов для поддержки разработки проекта и изготовления программного обеспечения (см. 5.5);

d) установление требований к программному обеспечению и целей обеспечения надежности (см. 5.6, приложение В);

e) классификация неисправностей программного обеспечения (см. 5.7) и определение соответствующих параметров программного обеспечения (см. 6.2, приложение Е) для реализации стратегии обеспечения надежности программного обеспечения (см. 5.8);

f) применение соответствующих методов надежности при проектировании и изготовлении программного обеспечения (см. 6.1, 6.3);

g) инициирование повышения надежности, если применимо, с учетом различных ограничений при адаптации проекта (см. 6.4, 7.2);

h) мониторинг процессов разработки и изготовления для управления и получения обратной связи, необходимых для поддержания работоспособности программного обеспечения и обеспечения надежности системы в эксплуатации (см. раздел 7).

5.3 Действия жизненного цикла программного обеспечения

Жизненный цикл программного обеспечения включает в себя:

- определение требований системы одновременно к элементам аппаратного и программного обеспечения в соответствии с потребностями пользователей и ограничениями в применении системы;

- анализ требований определяет возможные варианты проекта и преобразует требования системы к сервисным приложениям в технические требования к проектам аппаратных и программных подсистем и разработке системы;

- проектирование структуры обеспечивает решение для выполнения требований системы, путем распределения элементов системы по блокам подсистем, чтобы установить базовую структуру для декомпозиции программных подсистем и определения соответствующих функций программного обеспечения в соответствии с установленными требованиями;

- детальное проектирование обеспечивает проект каждой идентифицированной функции в структуре системы и создает необходимые блоки и интерфейсы программного обеспечения для функции, которая может быть распределена между программным и/или аппаратным обеспечением. Функции, отнесенные к программному обеспечению, должны быть определены достаточно подробно для кодирования и тестирования. Функцию программного обеспечения можно обозначить как программную подсистему и идентифицировать как объект конфигурации программного обеспечения для управления проектированием;

- реализация включает изготовление программных блоков, которые соответствуют критериям проверки и требованиям проекта, включая действия более низкого уровня:

- кодирование программных блоков;

- модульное тестирование для проверки программного блока на соответствие требованиям проекта;

- проверку подсистемы для верификации функций программного обеспечения на соответствие требованиям проекта;

- интеграцию, в процессе которой собирают программные блоки и подсистемы в соответствии с конфигурацией структуры проекта и устанавливают программную систему в аппаратную систему для тестирования;

- приемку, в процессе которой устанавливают возможности системы и валидируют программные приложения по предоставлению требуемых услуг системы в целевых условиях; приемочные тесты программного обеспечения включают действия более низкого уровня:

- испытания на повышение надежности для повышения безотказности программного обеспечения системы проводят после того, как система программного обеспечения полностью интегрирована и выполнена, в смоделированных условиях эксплуатации, представляющих целевые условия;

- квалификационные испытания для подтверждения приемки системы программного обеспечения и поставки ее заказчику;
- эксплуатацию и сопровождение программного обеспечения, которые поддерживают работоспособность системы и отвечают за предоставление по запросам конкретных эксплуатационных услуг;
- обновление/улучшение программного обеспечения, направленное на улучшение работы программного обеспечения за счет расширения его свойств;
- распоряжение программным обеспечением и прекращение поддержки конкретной услуги программного обеспечения.

В приложении В представлены типичные требования к системе программного обеспечения и соответствующие действия по обеспечению надежности на стадиях жизненного цикла программного обеспечения.

На рисунке 1 показаны основные действия по обеспечению надежности, важные для жизненного цикла выполнения проекта программного обеспечения.

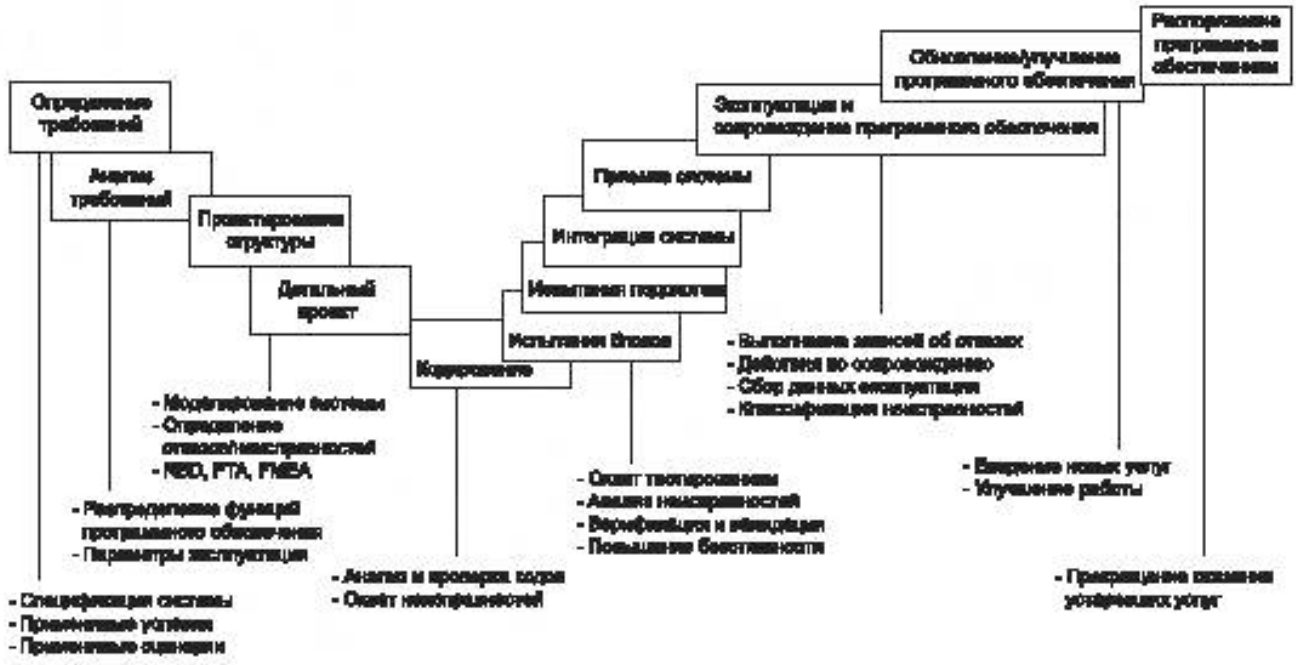

Рисунок 1 — Действия жизненного цикла программного обеспечения

5.4 Свойства надежности программного обеспечения

Свойства надежности программного обеспечения характеризуют параметры и показатели программного обеспечения. Обеспеченные проектом особые свойства, связанные с применением, необходимо учитывать при включении их в проект и структуру системы для достижения комплексных целей надежности аппаратно-программного комплекса.

Основные свойства надежности программного обеспечения или неотъемлемые характеристики надежности программного обеспечения, способствующие достижению целей надежности системы, включают (но не ограничиваются этим):

- готовность: подготовленность к работе программного обеспечения;
- безотказность: непрерывность оказания услуг программным обеспечением;
- ремонтопригодность: простота модификации, обновления и улучшения программного обеспечения;
- восстанавливаемость: восстановление программного обеспечения после отказа с внешними действиями или без них;
- целостность: корректность данных программного обеспечения.

Свойства, относящиеся к конкретному применению системы, способствующие достижению целей надежности системы, включают (но не ограничиваются):

- защищенность: защита от вторжения при применении и использовании программного обеспечения;
- безвредность: для предотвращения вреда при применении и использовании программного обеспечения;
- работоспособность: робастность, отказоустойчивость и бесперебойная работа;
- возможность многократного использования: использование существующего программного обеспечения для других целей;
- поддержку: поддержание работы системы с помощью ресурсов логистики и обслуживания;
- переносимость: кроссплатформенные применения.

Эти свойства надежности, присущие программному обеспечению и его применению, связаны с особенностями его работы, предусмотренными проектом и применением системы программного обеспечения.

5.5 Условия разработки программного обеспечения

Целью менеджмента надежности является обеспечение сбалансированных условий проектирования для творчества в рамках ресурсов бюджета, графика и поставленных целей проекта. Организации, связанные с разработкой программного обеспечения и предоставлением программных услуг, ориентированы на пользовательские приложения. Адаптация проектов программного обеспечения необходима для управления распределением доступных ресурсов и поиска подходящих вариантов проекта для его эффективного выполнения. Выбор и принятие применимых процессов для обеспечения надежности конкретной системы программного обеспечения осуществляют с помощью процесса адаптации проекта. Рекомендуемые задачи процесса адаптации приведены в 7.2. Следует изучить возможности аутсорсинга, повторного использования программного обеспечения и применения готовых коммерческих программных продуктов (COTS) для интеграции системы при проектировании.

Условия разработки программного обеспечения опираются на организованный процесс продвижения передовых методов проектирования для безошибочного кодирования, минимизации ошибок при определении требований и обеспечении верификации испытаний при выпуске программного обеспечения. Культурные аспекты в управлении программным обеспечением часто принимают форму концепции модели зрелости возможностей для разработки инфраструктуры [8]. Это похоже на формальную реализацию интеграции модели зрелости [9], описанную в приложении С для управления процессами программного обеспечения. Разработка программного обеспечения — это технический процесс в соответствии с установленными правилами разработки программного обеспечения и рекомендациями по его применению. Условия и принципы разработки программного обеспечения должны быть включены в политику организации, по установлению его целей и задач достижения надежности.

При разработке программного обеспечения часто используют приемы CASE (автоматизированной разработки программного обеспечения). Эффективная автоматизированная система обеспечивает точность вычислений, отслеживаемость данных, управление конфигурацией и средства сбора необходимых результатов измерений или входных показателей для автоматизированных моделей. Большая часть систем сбора данных для отчетов об отказах при эксплуатации, анализа и корректирующих действий автоматизированы. Существующие данные о программных продуктах и услугах являются незаменимыми и ценными.

5.6 Установление требований к программному обеспечению и целей в области надежности

Для стадий жизненного цикла программного обеспечения должны быть установлены требования к программному обеспечению. Для каждой стадии должны быть определены применимые действия по обеспечению надежности. Установление сроков выполнения действий по обеспечению надежности очень важны. Применение действий по обеспечению надежности зависит от времени и оказывает значительное влияние на стоимость жизненного цикла системы [10]. Адаптация проекта необходима для поиска компромиссных вариантов проектов с учетом ограничений. Требования к программному обеспечению и цели обеспечения в области надежности должны быть частью общих спецификаций на программный продукт. Стратегия реализации надежности программного обеспечения описана в 5.8. Методология обеспечения надежности в программных модулях или блоках при построении структуры системы программного обеспечения рассмотрена в разделе 6. Процесс адаптации описан в 7.2.

Действия по обеспечению надежности, связанные с требованиями к программному обеспечению, зависят от конкретного его применения. Они отражают требования к проекту и изготовлению програм-

много обеспечения с учетом необходимых функций системы и оказания услуг при его применении. Систематические подходы к реализации соответствующих действий по обеспечению надежности в течение всего жизненного цикла программного обеспечения гарантируют выполнение целей в области надежности. Конкретные цели в области надежности определяют на основе выбора ключевых свойств надежности и соответствующих количественных показателей. Требования к надежности программного обеспечения могут быть сформулированы для конкретных проектов с использованием базовой информации, приведенной в приложении В.

Условия, влияющие на требования к надежности системы, состоящей из аппаратных и программных средств, описаны в спецификациях надежности системы [11]. Необходимо учитывать следующие факторы, влияющие на достижение надежности при разработке программного обеспечения:

- культуру проектирования организации, процесс развития возможностей и опыт проектирования, разработки и изготовления программного обеспечения (см. приложение С);
- понимание условий применения, потребностей пользователей и изменения динамики рынка при разработке новой платформы или функции для практического изготовления;
- процессы документирования, такие как записи об отказах, сбор данных, управление конфигурацией для контроля версий программного обеспечения и ведение записей об экспериментальных данных;
- применение правил проектирования программного обеспечения для предотвращения неисправностей путем управления процессом проектирования для оптимизации изготовления программного обеспечения в соответствии со сложностью системы, программ и функций;
- эффективное использование применимых методов и приемов программирования, таких как структурированное проектирование, обеспечение устойчивости к неисправностям, документальный анализ проекта [12] и управление неисправностями программного обеспечения, для повышения его надежности;
- выбор соответствующих языков программирования более высокого порядка, подходящих для структурированной разработки конкретного программного обеспечения;
- установление требований к квалификации и оценке показателей надежности программного обеспечения.

5.7 Классификация неисправностей программного обеспечения

Неисправности программного обеспечения могут быть классифицированы как ошибки в спецификации, ошибки в проекте, ошибки программирования, ошибки, введенные при компиляции, или неисправности, возникшие при сопровождении программного обеспечения.

Классификация неисправностей программных средств обеспечивает способ сбора и группировки соответствующей информации о неисправностях программных средств. Процесс классификации неисправностей помогает разработчикам программного обеспечения выявить необычные неисправности и предусмотреть корректирующие действия. Цель состоит в устранении повторения аналогичных неисправностей.

Классификация ортогональных дефектов (ODC) [13] — это метод, используемый при разработке программного обеспечения для анализа данных о неисправностях (дефектах) программного обеспечения. ODC направлена на анализ проблем качества программного обеспечения, касающихся его проектирования и кодирования в процедурной языковой среде. Дефект — это невыполнение требования, связанного с предполагаемым или установленным использованием программного обеспечения. В настоящем стандарте неисправность, вызванная неспособностью программного обеспечения выполнять требуемые функции, демонстрирует характеристики атрибута дефекта в схеме ODC. Атрибут дефекта — это описание, содержащее соответствующую информацию, связанную с неисправностью программного обеспечения. Метод ODC собирает информацию о неисправности программного обеспечения в виде дефекта для анализа и моделирования. Анализ данных ODC представляет собой ценный метод диагностики для оценки зрелости программного продукта на различных стадиях жизненного цикла программного обеспечения. ODC также можно использовать для оценки процесса путем анализа типов триггеров и определения конкретных технических потребностей для стимулирования отсутствующих триггеров. Данные анализа причин неисправностей (дефектов) представляет собой способ снижения количества неисправностей программного обеспечения и повышения его надежности.

Атрибуты дефекта ODC включают действия, триггер, цель, тип дефекта, классификатор дефекта, источник, влияние и возраст. Их обычно собирают и анализируют во время разработки программного обеспечения для улучшения проекта. Информация о дефектах доступна в два конкретных момента

времени. При обнаружении неисправности обычно известны обстоятельства, приводящие к выявлению неисправности и вероятному воздействию на пользователя. После того как неисправность устранена, суть и место неисправности зафиксированы. Категории ODC охватывают семантику неисправности (дефекта) с этих двух точек зрения. С помощью определения действий во время процесса разработки и сопоставления их с триггерами ODC, ODC обеспечивает понимание и позволяет получить информацию о неисправностях (дефектах) для организации разработки программного обеспечения.

Набор атрибутов дефекта суммируют: a) когда обнаружена неисправность, это называют открытием, и b) когда неисправность устранена, это называют закрытием. ODC наиболее полезен в зрелых организациях, занимающихся разработкой программного обеспечения, где обычно собирают и анализируют обширные данные, позволяющие улучшить программное средство.

В приложении D представлена классификация атрибутов дефектов программного обеспечения.

5.8 Стратегия обеспечения надежности программного обеспечения

5.8.1 Предотвращение неисправностей программного обеспечения

Коды программного обеспечения генерируют при изготовлении программного средства. Неисправность, допущенная при разработке и кодировании программного обеспечения, может проявиться при запуске и стать неисправностью программного обеспечения, приводящей к отказу системы. Поскольку неисправности являются основной причиной отказов системы, предотвращение отказов, возникших в результате проектирования, и такие действия, как проверка кодов и удаление неисправностей, которые не были обнаружены при тестировании, являются распространенным способом уменьшения количества неисправностей в течение жизненного цикла программного обеспечения. Рекомендуемая стратегия предотвращения неисправностей программного обеспечения включает предотвращение и устранение неисправностей.

a) Предотвращение неисправностей:

- установление цели предотвращения неисправностей при разработке программного обеспечения;
- инициирование анализа спецификаций требований;
- проведение раннего взаимодействия с пользователем и уточнение требований к программному обеспечению;
- введение формальных методов, где это применимо и практически осуществимо;
- внедрение систематических методов повторного использования программного обеспечения и гарантии его применения.

b) Устранение неисправностей:

- обнаружение и устранение существующих неисправностей программного обеспечения путем тестирования (испытаний);
- проведение документальной проверки по выявлению неисправностей, их исправления и проверки результатов исправлений;
- выполнение корректирующего и предупреждающего обслуживания при эксплуатации программного обеспечения.

5.8.2 Контроль неисправностей программного обеспечения

Неисправности программного обеспечения трудно обнаружить, их устранение может быть достигнуто различными способами, включая тщательное тестирование и проверку программного обеспечения. Исчерпывающее тестирование программного обеспечения часто ограничено ресурсами времени и средств. Однако одно только тестирование не обеспечивает полного устранения неисправности. При контроле неисправностей программного обеспечения используют методы обеспечения устойчивости и прогнозирования, чтобы минимизировать проявление скрытых неисправностей программного обеспечения или неисправностей, которые все еще могут существовать после выпуска программного обеспечения и при его использовании. Рекомендуемая стратегия контроля неисправностей программного обеспечения включает в себя устойчивость к неисправностям и прогнозирование неисправностей (отказов).

a) Обеспечение устойчивости к неисправностям:

- разработка методологии локализации, обнаружения и устранения неисправностей;
- выполнение различных вариантов проекта программного обеспечения и схем снижения скорости передачи данных;
- введение многовариантных методов программирования;
- внедрение методов самопроверки программирования.

b) Прогнозирование неисправностей (отказов):
- установление взаимосвязей неисправностей (отказов) в операционной среде;
- создание системы сбора данных;
- проведение испытаний на повышение надежности, где это применимо;
- разработка и внедрение соответствующих моделей безотказности для оценки неисправностей (отказов);
- уточнение методов прогнозирования времени проектирования и выпуска версии программного обеспечения.

6 Методы обеспечения надежности при применении программного обеспечения

6.1 Практика разработки программного обеспечения для достижения надежности

Зрелость возможностей в разработке программного обеспечения отражает способность организации разрабатывать непротиворечивое и надежное программное обеспечение для намеченного применения. Следующие методы предотвращения и контроля неисправностей рекомендуются для включения, где это применимо, в разработку программного обеспечения:

a) стандартизированные методы проектирования структуры высокого уровня, детального проектирования, кодирования и тестирования, а также документирования для облегчения обмена информацией и предотвращения неисправностей;

b) разработка модульных проектов при проектировании программных блоков и подсистем с четко определенными программными функциями и интерфейсами путем создания простых, отдельных и независимых программных блоков для облегчения взаимодействия при проектировании, обслуживании, прослеживании неисправностей, уменьшения количества неисправностей и устранения неисправностей и ошибок;

c) использование шаблонов при проектировании, которые являются общими повторно используемыми решениями хорошо протестированного программного обеспечения, в качестве шаблонов для решения задач, позволяющих ускорить процесс разработки программного обеспечения;

d) введение методов проектирования в соответствии с установленными правилами, где это уместно, для контроля и документирования процесса проектирования и разработки программного обеспечения;

e) использование методов разработки надежного программного обеспечения [14] для оценки и повышения надежности программного обеспечения [15];

f) повторное использование программного обеспечения, имеющегося в библиотеке программного обеспечения, состоящего из хорошо протестированных блоков и подсистем для аналогичного применения, снижения затрат, сокращения времени разработки и минимизации введения новых неисправностей при проектировании;

g) разработка методов регрессионного тестирования и испытаний для обеспечения функциональности существующего программного обеспечения по мере введения новых функций или устранения неисправностей;

h) тестирование программных блоков и подсистем для верификации функций на низком уровне проекта и валидация работы интегрированной системы со структурой высокого уровня для постепенного устранения ошибок и предотвращения распространения неисправностей;

i) проведение контроля и анализа требований к программному обеспечению, кодов программного обеспечения, руководства пользователя, учебных материалов и тестовых документов для выявления и устранения как можно большего количества возможных ошибок, использование, где это практически возможно, различных групп анализа для сопоставления результатов;

j) управление изменениями для уменьшения количества неисправностей, в том числе выполнение процесса управления версиями и изменениями при управлении конфигурацией программного обеспечения;

k) анализ первопричин проблем и выполнение соответствующих корректирующих действий для постоянного улучшения программного обеспечения;

l) создание системы сбора данных для базы знаний, включающей данные о неисправностях программного обеспечения и данные о его работе.

6.2 Показатели надежности программного обеспечения и сбор данных

Показатели надежности программного обеспечения являются показателями свойств надежности системы программного обеспечения. Система показателей обеспечивает количественную шкалу и метод определения значения показателя для конкретного программного обеспечения. Приведенные отраслевые стандартные показатели получены либо путем непосредственного определения, либо путем вывода. Их используют в качестве показателей работы системы программного обеспечения. Следующие показатели программного обеспечения являются фактически отраслевыми стандартами для применения. Их следует учитывать при необходимости для оценки надежности систем программного обеспечения.

a) Коэффициент готовности: вероятность работоспособного состояния в конкретный момент времени в течение периода работы системы.

b) Частота отказов: количество отказов за время работы системы.

c) Наработка до отказа: продолжительность безотказной работы.

d) Время восстановления: продолжительность периода восстановления системы из неисправного состояния (неработоспособного состояния) в состояние нормальной работы (работоспособное состояние).

e) Плотность отказов: количество неисправностей, содержащихся в тысяче строк исходного кода (KSLOC) или в функциональной точке, показатель используют для оценки безотказности программного обеспечения.

f) Функциональная точка: функциональный размер применения программного обеспечения для планирования проекта программного обеспечения с помощью метода анализа функциональных точек [16].

g) Охват кодов тестированием: степень, в которой исходные коды и логические ветви программы могут быть систематически протестированы. Охват кодов является индикатором тщательности тестирования программного обеспечения, показатель используют для представления охвата неисправностей, который указывает процент неисправностей, обнаруженных во время тестирования при выполнении кода.

h) Интенсивность устранения неисправностей: количество неисправностей, обнаруженных и исправленных в программном средстве для определения периода времени или продолжительности изготовления программного обеспечения. Интенсивность устранения неисправностей используют в анализе повышения надежности для определения тенденции повышения безотказности.

i) Остаточные неисправности программного обеспечения: оценка количества ошибок, которые все еще остаются в программном продукте после тестирования.

j) Время выпуска программного обеспечения: оценка времени выпуска программного продукта в соответствии с графиком, на основе установленных критериев с приемлемым уровнем ошибок, оставшихся в программном средстве для управления проектом программного обеспечения.

k) Сложность программного обеспечения: степень сложности при разработке и изготовлении функций или системы программного обеспечения; другие меры сложности, основанные на концепции сложности, включающие сложность программы, функциональную сложность, сложность эксплуатации; меры, связанные со сложностью, используют в качестве входных данных при оценке безотказности и в моделях прогнозирования.

В течение жизненного цикла программного обеспечения используют много показателей для различных целей. Все параметры и показатели программного обеспечения могут быть сгруппированы в три основные категории для облегчения сбора данных.

1) Данные о неисправностях: сбор информации о проблемах программного обеспечения для определения воздействия неисправностей и эффективности процесса записей для улучшения обслуживания программного обеспечения.

2) Данные о продукте: сбор информации о программном обеспечении с указанием объема его функций, сложности, местоположения при использовании и других характеристик для облегчения использования опытных данных в качестве входных данных для получения преимуществ при разработке нового продукта. Данные включают сведения об истории работы программного обеспечения, а также информацию о других группах программных продуктов.

3) Данные о процессе: сбор информации о процессе восстановления программного обеспечения и условиях в момент обнаружения и устранения неисправностей для входных данных модели безотказности при прогнозировании безотказности.

Процесс сбора данных имеет важное значение для определения показателей надежности программного обеспечения и характеристик его работы. Эффективная система сбора данных должна быть практичной при выполнении. Объем и типы данных должны быть относительно простыми для сбора, легко интерпретируемыми для анализа и полезными для оценки обеспечения и улучшения надежности программного обеспечения. Собранные данные используют для определения тенденций изменения безотказности системы, частоты и продолжительности времени, необходимого для обслуживания программного обеспечения, времени отклика на сервисные вызовы, требований к восстановлению ухудшенной работы и технической поддержке.

В приложении Е представлены примеры данных о программном обеспечении, полученных при сборе данных.

6.3 Оценка надежности программного обеспечения

6.3.1 Процесс оценки надежности программного обеспечения

Целью процесса обеспечения надежности программного обеспечения является обеспечение зрелости системы программного обеспечения при разработке и достижение необходимой надежности. Процесс оценки обеспечивает верификацию требований к программному обеспечению и валидацию надежности программного обеспечения после изготовления системы. Процесс оценки надежности включает важнейшие действия по разработке программного обеспечения, принятые из устоявшихся отраслевых практик [14]. Рекомендуется следующий процесс выполнения оценки надежности программного обеспечения:

- определение потребностей пользователей, цели работы системы и разработка спецификации надежности;
- установление профиля эксплуатации программного обеспечения;
- распределение применимых показателей надежности;
- выполнение анализа и оценки надежности для определения вариантов и возможных решений;
- проведение тестирования (испытаний) программного обеспечения и его параметров;
- проведение верификации программного обеспечения и валидации системы программного обеспечения;
- представление данных о повышении надежности программного обеспечения и прогнозируемых тенденций ее изменения;
- сопоставление результатов оценки с данными обратной связи.

Действия по оценке надежности программного обеспечения описаны в следующих пунктах.

6.3.2 Спецификация работы системы и ее надежности

Цель состоит в определении цели работы системы для разработки спецификации надежности системы [11], если она не представлена заказчиком или пользователем. Рекомендуется следующий процесс:

- определение сценариев работы системы и условий ее применения;
- определение соответствующих факторов, влияющих на работу системы;
- определение границ системы и интерфейсов с внешними взаимодействующими системами;
- определение соответствующих параметров работы системы;
- определение структуры системы, конфигурации аппаратных и программных средств;
- определение взаимодействующих аппаратных и программных функций системы;
- описание и количественное определение показателей надежности соответствующих аппаратных и программных функций, включая коэффициент готовности, вероятность безотказной работы и возможность восстановления, связанные с критериями поддержки технического обслуживания.

Документация спецификации надежности системы должна включать в себя следующие данные как часть спецификации системы:

- идентификацию системы;
- цель работы системы;
- профиль эксплуатации системы;
- цели надежности системы при эксплуатации;
- конфигурацию системы;
- функции системы;
- требования к надежности для каждой функции;
- требования к сопровождению системы.

Для системы программного обеспечения важно рассмотреть профиль эксплуатации, который влияет на время выполнения функций программного обеспечения по запросу. Для представления аппаратной и программной системы может быть построена структурная схема надежности функций [17]. Созданные функциональные блоки способствуют распределению соответствующих показателей надежности программного обеспечения по функциям программного обеспечения в соответствии с установленной структурой системы программного обеспечения и конфигурацией программного обеспечения.

6.3.3 Установление профиля эксплуатации программного обеспечения

Профиль эксплуатации — это последовательность необходимых действий, которые должны быть выполнены объединенной аппаратно-программной системой для достижения ее целей или задач в области оказания услуг. Работа системы в большой степени зависит от условий, в которых она работает. Условия могут влиять на физические изменения оборудования, но не могут влиять на функции программного обеспечения, предоставляемые системой при ее эксплуатации.

Разработка профиля эксплуатации — это количественная характеристика того, как следует использовать программное обеспечение. Эксплуатационные данные и соответствующую информацию обычно собирают путем опроса потребителей и сбора данных об оказании услуг при эксплуатации системы. Ниже приведены рекомендуемые процессы для разработки профиля эксплуатации:

a) определение профиля потребителя или заказчика, намеренных приобрести систему программного обеспечения, с установлением их потребностей и типов, таких как организация или частное лицо;

b) установление профиля пользователя для разных типов пользователей, таких как человек или сотрудник организации, или взаимодействующих систем программного обеспечения, работающих или использующих систему программного обеспечения для конкретного применения;

c) определение профиля режимов системы, в которых она работает, и последовательности или порядка режимов работы системы, таких как тестирование программного обеспечения для обновления ее обслуживания, или обычная пакетная обработка данных при изготовлении системы программного обеспечения;

d) определение функционального профиля путем оценки каждого режима функций работы и функций услуг, таких как создание сообщения электронной почты или поиск адреса в соответствии с функциональными требованиями программного обеспечения;

e) определение профиля эксплуатации на основе функциональных профилей, установленных для работы системы;

f) определение профиля информации путем сбора данных о применении программного обеспечения в соответствии с жизненным циклом разработки программного обеспечения.

Функциональный профиль представляет собой возможности системы с точки зрения пользователя. С точки зрения разработчика, функциональный профиль представляет собой операции системы, которые реализуют требуемые функции. С точки зрения надежности, профиль эксплуатации представляет собой набор различных сценариев работы системы и вероятности их возникновения. Профиль эксплуатации обеспечивает необходимые входные данные для разработки тестовых примеров при моделировании операций системы программного обеспечения в эксплуатации и конкретных применений функциональных свойств программного обеспечения при использовании. Выполнение тестовых примеров при тестировании программного обеспечения обеспечивает ценную информацию и сбор данных для оценки вероятности безотказной работы и валидации результативности выполнения обслуживания программного обеспечения в условиях эксплуатации.

6.3.4 Распределение показателей надежности

Распределение показателей надежности в системе программного обеспечения основано на концепции моделирования функций структуры системы, чтобы отразить требования целей надежности системы. Первоначальное назначение применимых показателей надежности, таких как вероятность безотказной работы и коэффициент готовности, основано на экспериментальных данных. Значения таких показателей уточняют в процессе итеративного анализа и оценки. Распределение значений вероятности безотказной работы и коэффициента готовности по различным подсистемам программного обеспечения и функциональным блокам выполняют в соответствии с их сложностью, критичностью, достижимыми целевыми значениями вероятности безотказной работы и коэффициента готовности и другими факторами, влияющими на процесс распределения.

Разработка модели системы для программного обеспечения существенно отличается от разработки модели аппаратного обеспечения из-за присущих им параметров эксплуатации. Для каждого режима эксплуатации системы, который включает в себя функции программного обеспечения в качестве

элементов конфигурации, составляют свой набор составляющих программных блоков. Каждый режим имеет уникальное время применения, связанное с продолжительностью выполнения программного блока по запросу при эксплуатации системы. Это указывает продолжительность каждого режима работы системы. Моделирование системы программного обеспечения включает в себя количество строк исходного кода в каждом программном блоке, сложность кода и другую информацию, относящуюся к ресурсам разработки программного обеспечения, таким как язык программирования и условия проектирования. Их используют для установления начальной интенсивности отказов для прогнозирования вероятности безотказной работы или коэффициента готовности объектов конфигурации программного обеспечения.

6.3.5 Анализ и оценка надежности

Следующие действия по анализу и оценке надежности необходимы для поддержки разработки систем программного обеспечения. Процесс является итеративным для оптимизации требований к проектированию надежности в соответствии с целями работы системы. Моделирование коэффициента готовности и вероятности безотказной работы используют для анализа и оценки выполнения функций программного обеспечения, зависящих от времени.

а) Моделирование коэффициента готовности и вероятности безотказной работы функций.

Простой подход к анализу коэффициента готовности и вероятности безотказной работы системы, состоящей из аппаратного и программного обеспечения, заключается в формировании структурной модели системы. Функциональная модель коэффициента готовности и вероятности безотказной работы для комбинированной аппаратно-программной системы, состоящей из функциональных блоков, может быть построена с использованием метода структурной схемы надежности (RBD) [17]. Модель выполняет декомпозицию системы на отдельные модели подсистем, представляющие составляющие аппаратные и программные элементы системы: анализ дерева неисправностей (FTA) [18], цепи Маркова [19] и сети Петри [20] также полезны для разработки моделей коэффициента готовности и вероятности безотказной работы системы. Например, FTA может быть использован для моделирования надежности системы с помощью динамических вентилей, что позволяет определить функции коэффициента готовности и вероятности безотказной работы аппаратного и программного обеспечения для поиска компромиссных решений и улучшений [21]. Следует отметить, что RBD и FTA логически эквивалентны. RBD ориентирован на успех системы, а FTA — на отказ системы.

Модель коэффициента готовности и вероятности безотказной работы аппаратной подсистемы состоит из всех аппаратных элементов системы с функциональными блоками, построенными соответствующим образом для представления соответствующей структуры конфигурации резервирования аппаратной подсистемы. Это должно облегчить прогнозирование интенсивности отказов отдельных аппаратных компонентов и определение коэффициента готовности или вероятности безотказной работы подсистемы аппаратных средств в соответствии с методами прогнозирования. Возможна ситуация, когда одна или несколько аппаратных подсистем обслуживают различные функции конфигурации системы.

Модель вероятности безотказной работы подсистемы программного обеспечения строят с использованием программных модулей в качестве блоков, обеспечивающих выполнение функций программного обеспечения. Программный модуль — это самый низкий уровень конфигурации объекта программного обеспечения. Отказы программных модулей не являются независимыми, как в случае элементов аппаратных средств. Программные коды — это виртуальные объекты, не подверженные физическим изменениям. Программные блоки отказывают в соответствии с профилем эксплуатации системы, что влияет на схему конфигурации модели безотказности программного обеспечения. Моделирование вероятности безотказной работы программного обеспечения необходимо для использования информации профиля эксплуатации при разработке структуры конфигурации программного обеспечения. Программа подсистемы программного обеспечения может состоять из одного или нескольких компонентов программных блоков для выполнения требуемых функций. Программа подсистемы программного обеспечения, находящаяся в подсистеме аппаратного обеспечения, сформирована в виде объекта конфигурации программного обеспечения. Взаимодействие и взаимозависимость подсистемы программного обеспечения и спроектированный для нее объект аппаратного обеспечения необходимы для предоставления конкретных функций подсистемы программного обеспечения при работе системы. Может быть несколько комбинированных программных и аппаратных подсистем, обслуживающих различные функции конфигурации системы. В приложении F приведен пример, иллюстрирующий взаимодействие вероятности безотказной работы функций аппаратно-программного обеспечения и определения интенсивности отказов системы для оценки ее вероятности безотказной работы.

Оценка комбинированной аппаратно-программной системы должна сначала установить взаимодействия функций безотказности аппаратного и программного обеспечения для вывода коэффициента готовности функций системы. Общая продолжительность неработоспособного состояния системы или время восстановления за время работы системы необходимы для оценки коэффициента готовности системы.

b) Определение вероятности безотказной работы программных функций.

Отказы программного обеспечения могут возникать при эксплуатации системы. Определение интенсивности отказов программного обеспечения для использования в моделировании требует рассматривать программное обеспечение как подсистему, которая находится в подсистеме аппаратного средства, сформированной в качестве объекта конфигурации программного обеспечения. Подсистема программного обеспечения может выполнять одну или несколько функций. С точки зрения конечного пользователя функция — это способность системы оказывать требуемую услугу. Эта функцию может быть выполнена с помощью объекта конфигурации программного обеспечения, так что функцию требуемой услуги распознает система контроля версий. Блок программного обеспечения, предназначенный для выполнения только одной функции, может быть объектом конфигурации. Программа подсистемы программного обеспечения, требующая, чтобы несколько программных модулей выполняли одну функцию, также может быть элементом конфигурации. Подсистема программного обеспечения может состоять из нескольких программ для предоставления набора связанных функций. Каждая из этих программ является объектом конфигурации по определению. Концепцию объекта конфигурации программного обеспечения рассматривают с точки зрения управления версиями проектов программного обеспечения. Объект конфигурации программного обеспечения необходим для прослеживания изменений проекта. Каждому изменению проекта присваивают номер версии для идентификации. Ссылка на версию программного обеспечения необходима для отслеживания результативности обновления обслуживания программного обеспечения. Тенденцию повышения надежности выявляют с помощью показателей улучшения работы системы, когда система работает с новой версией, заменяющей старую. Обеспечение требуемых функций при эксплуатации системы является стимулом соответствия целям надежности системы.

Функции программного обеспечения, составляющие систему, относятся к синхронизации и топологии безотказности.

Конфигурация синхронизации является проблемой, когда различные функции активны или пассивны в течение определенного периода времени работы системы. Основные временные взаимосвязи функций программного обеспечения являются параллельными или последовательными. Функции являются параллельными, если они действуют одновременно. Функции являются последовательными, если они действуют одна за другой. Возможно также, что время выполнения функций частично совпадает, что приводит к гибридной параллельно-последовательной конфигурации синхронизации. Параллельные функции программного обеспечения имеются в системах, которые обслуживают несколько центральных процессоров, например в многопроцессорной системе или распределенной системе. Ссылки на время выполнения определяют как время выполнения, время работы системы и календарное время.

Топология безотказности связана с количеством функций в системе, которые могут отказать до отказа системы. Топология безотказности — это связь отказа отдельной функции с отказом системы в целом. Функции программного обеспечения обычно связаны в последовательную топологию. Отказ одной функции может привести к отказу системы программного обеспечения. Проект программного обеспечения, устойчивый к неисправностям, может быть использован для защиты системы в случае отказа одной или нескольких функций.

c) Опорное время выполнения функции программного обеспечения.

Интенсивность отказов программного обеспечения может быть выражена относительно трех разных параметров времени.

- Время выполнения — это время выполнения центрального процессора, которое накапливается, когда программа выполняет инструкции. Время выполнения используют для определения интенсивности отказов, связанных со временем применения подсистемы программного обеспечения.

- Время работы системы, которое увеличивается всякий раз, когда работает система аппаратного программного обеспечения в целом. Это используют для определения интенсивности отказов, связанных со временем эксплуатации программного обеспечения подсистемы при непрерывной работе.

- Календарное время — это период времени, используемый для планирования и составления графика выполнения проекта. Календарное время всегда больше первых двух параметров.

Во время эксплуатации системы программы не всегда работают непрерывно. Некоторые программы могут одновременно использовать время одного центрального процессора. Может также существовать несколько блоков центральных процессоров, что допускает наложение выполнения программ. Интенсивности отказов различных программ необходимо объединить, чтобы получить общую интенсивность отказов программного обеспечения. Их преобразуют в общую рамку опорного времени эксплуатации системы. Это тот же период времени, который использован для выражения интенсивностей отказов оборудования для облегчения определения интенсивности отказов объединенной аппаратно-программной системы.

d) Критичность функции программного обеспечения.

Функции программного обеспечения часто используют для управления критической системой, где отказ может привести к катастрофическим последствиям. Критичность функций программного обеспечения должна быть определена на ранних этапах определения концепции системы и оценена в процессе проектирования программного обеспечения системы.

Критичность отказов функций должна быть классифицирована в спецификациях системы, как критическая, серьезная или второстепенная, на основании установленных критериев, и подтверждена анализом показателей надежности системы.

Уровень риска, связанного с критической функцией программного обеспечения, может быть определен и оценен с помощью методов оценки риска. Менеджмент риска проекта [22] должен быть направлен на предотвращение неисправностей и обеспечение устойчивости к неисправностям в ситуациях, где можно снизить тяжесть последствий отказов.

Анализ дерева неисправностей [18] можно использовать для выявления возможных причин нежелательной вершины событий. FTA используют для исследования возможных неисправностей и их причин, а также для количественной оценки их вклада в коэффициент неготовности системы. Анализ дерева неисправностей — это нисходящий технический подход, где отправной точкой является программа подсистемы высшего уровня, которая следует по иерархической структуре программного обеспечения до самого низкого программного модуля. Для возможных неисправностей могут быть индивидуально идентифицированы и оценены соответствующие вероятности возникновения неисправностей. Количественная оценка обеспечивает признаки или величину критичности функции программного обеспечения. Это представляет интерес для оптимизации проекта и предотвращения неисправностей.

Анализ видов и последствий отказов [23] может быть использован для определения возможных видов отказов и неисправностей в блоках программного обеспечения и их влияния на следующую подсистему более высокого уровня иерархической структуры программного обеспечения. Анализ видов и последствий отказов — это технический подход снизу вверх. Его можно расширить и использовать для анализа критичности функций программного обеспечения. Анализ критичности сочетает в себе количественную оценку вероятности возникновения отказов и качественную информацию о значимости отказов для поддержки компромиссных решений и снижения количества неисправностей.

Другие методы анализа надежности системы [24] используют для декомпозиции программного обеспечения и моделирования системы. Их можно выборочно использовать для подробной оценки показателей безотказности и ремонтопригодности функций программного обеспечения в комбинированных аппаратно-программных системах.

Уровень целостности программного обеспечения — это показатель, отражающий уникальные для проекта характеристики, которые определяют значимость программного обеспечения для пользователя. Примеры уникальных характеристик проекта включают сложность, критичность, риск, уровень безопасности, уровень защиты, желаемое выполнение и вероятность безотказной работы программного обеспечения. Уровень целостности программного обеспечения определяется классификацией критичности влияния последствий отказов и соответствующих частот возникновения отказов [25]. Критичность функций программного обеспечения также зависит от конкретного применения. Для систем, связанных с безопасностью, уровень целостности безопасности должен быть определен и включен в разработку системы программного обеспечения для соответствия требованиям функциональной безопасности [26]. Для систем, связанных с безопасностью, должны быть включены конкретные требования безопасности системы [27].

6.3.6 Верификация программного обеспечения и валидация системы программного обеспечения

Спецификация программного обеспечения имеет тенденцию быть намного более сложной, чем спецификация физических аппаратных систем, таких как машины и электрические/электронные системы. «Правильность» программного обеспечения имеет первостепенное значение. Процесс верифика-

ции [2] заключается в определении того, что требования к программному обеспечению являются полными и правильными в соответствии со стадиями жизненного цикла программного обеспечения. Процесс валидации [7] состоит в определении соответствия работы системы программного обеспечения и услуг требованиям заказчика/пользователя. Для выполнения процессов верификации и валидации необходимы соответствующие вспомогательные системы, такие как испытательное оборудование, средства и дополнительные ресурсы. Активирующая система не дает вклада в функции работы программного обеспечения или тестируемой системы во время ее работы на стадиях жизненного цикла.

a) Верификация программного обеспечения.

Процесс верификации программного обеспечения заключается в подтверждении того, что установленные требования к системе программного обеспечения выполнены. Рекомендуются следующие действия процесса верификации:

- определение стратегии верификации программного обеспечения;
- разработка плана верификации на основе требований к программному обеспечению;
- идентификация ограничений, в том числе ограничений, связанных с проектными решениями;
- обеспечение того, что подключающая система для проведения верификации доступна, а соответствующие устройства и ресурсы для тестирования (испытаний) подготовлены;
- проведение верификации для демонстрации соответствия установленным требованиям проекта;
- документирование результатов и данных верификации;
- анализ данных верификации для инициирования корректирующих действий.

b) Валидация системы программного обеспечения.

Процесс валидации системы программного обеспечения должен представить объективные свидетельства того, что работа системы соответствует требованиям заказчика/пользователя. Рекомендуются следующие действия процесса валидации:

- определение стратегии валидации услуг в условиях эксплуатации и достижения удовлетворенности заказчиков/пользователей;
- подготовка плана валидации;
- обеспечение того, что подключающая система доступна для валидации, а соответствующие устройства и ресурсы — для тестирования (испытаний);
- проведение валидации для демонстрации соответствия услуг требованиям заказчика/пользователя;
- документирование результатов и данных валидации;
- выполнение анализа, записей и отчета о данных валидации, соответствующих критериям, определенным в стратегии валидации.

6.3.7 Тестирование и определение параметров программного обеспечения

a) Общие положения по тестированию программного обеспечения.

Тестирование программного обеспечения — это процесс выполнения программы или набора закодированных инструкций с целью проверки функций программного обеспечения и обнаружения неисправностей. Цели тестирования программного обеспечения различны в зависимости от требований проекта, готовности программного продукта, статуса зрелости программного обеспечения и графика тестирования в процессе жизненного цикла программного обеспечения. При планировании тестирования программного обеспечения необходимо учитывать следующее.

Планирование испытаний имеет важное значение и должно быть документировано для описания целей процесса, процедур и ресурсов тестирования.

Тестирование программного обеспечения требует знаний, навыков и опыта тестирования. Хотя многие процедуры и приемы тестирования автоматизированы и широко используются, хорошие методы тестирования требуют навыков, опыта, интуиции и творческого подхода для достижения надежных результатов. Ведение протокола тестирования важно для обеспечения точности и прослеживаемости данных тестирования.

Тестирование — это больше, чем просто отладка программного обеспечения для обнаружения мест неисправностей и их устранение. Тестирование также используют при верификации и валидации, оценке показателей готовности и безотказности программного обеспечения.

Результативность и эффективность процесса тестирования являются критериями для методов тестирования на основе охвата. Автоматизация тестирования может сократить время тестирования программного обеспечения и снизить стоимость проекта. Выбор соответствующих методов тестирования,

затраты на обучение и сопровождение, связанные с приобретением методов тестирования, должны быть учтены.

Тестирование не обязательно является наиболее результативным средством улучшения качества программного обеспечения без выполнения соответствующих последующих действий. Следует рассмотреть альтернативные методы, такие как проверка и анализ кода.

Тестирование программного обеспечения является лишь частью процессов повышения надежности и улучшения программного обеспечения. Для достижения целей надежности необходимо объединение других усилий по достижению целей в области надежности.

Полное тестирование может быть невыполнимым и непрактичным, а часто и дорогостоящим. Сложность программного обеспечения влияет на степень полноты тестирования. Проблема сложности часто ограничивает возможность обнаружения и устранения неисправности при тестировании.

Скрытые неисправности программного обеспечения существуют в программном обеспечении после его выпуска в эксплуатацию. Прогнозирование вероятности безотказной работы программного обеспечения является способом оценки времени тестирования, необходимого для снижения остаточных неисправностей программного обеспечения до приемлемого значения до выпуска версии программного обеспечения.

Тестирование, выходящее за рамки модульного тестирования, необходимо выполнять отдельными группами тестирования, не зависящими от групп, разрабатывающих программное обеспечение.

b) Типы программного тестирования.

Ниже приведены типы тестирования (испытаний) программного обеспечения, выполняемые в течение жизненного цикла программного обеспечения.

Тестирование модуля: тестирование одного программного модуля, который может быть скомпилирован до его интеграции в программное обеспечение или подсистему. Программную модель тестируют для верификации того, что подробные спецификации проекта для модуля правильно реализованы.

Тестирование подсистемы: тестирование программного обеспечения подсистемы, состоящей из одного или нескольких программных модулей, в качестве объекта конфигурации программного обеспечения для верификации выполнения требований к функционированию подсистемы.

Тестирование интеграции: тестирование системы программного обеспечения на аппаратном средстве в целом, состоящем из интегрированных подсистем, для верификации функциональной работы, выявления проблем в программных и аппаратных интерфейсах и взаимодействиях аппаратного и программного обеспечения, а также для валидации показателей безотказности.

Испытания на повышение надежности: тестирование программного обеспечения в итеративном процессе для повышения вероятности безотказной работы путем испытаний до отказа, анализа отказов, выполнения корректирующих действий в существующей версии программного обеспечения для обновления и продолжение испытаний с новой версией программного обеспечения. Прекращают испытания на повышение надежности при достижении установленного целевого значения вероятности безотказной работы программного обеспечения.

Квалификационные испытания: испытания для демонстрации того, что программное обеспечение соответствует спецификациям при его интеграции в аппаратной системе и готовности к использованию в целевых условиях. Перед выпуском окончательной версии для распространения программного обеспечения и в целях обеспечения качества часто проводят альфа- и бета-тестирование. Альфа-тестирование является внутренним испытанием, проводимым разработчиком программного обеспечения перед его выпуском и передачей внешним пользователям. Бета-тестирование — это испытания в условиях эксплуатации, проводимые ограниченным числом пользователей в соответствии с предполагаемым применением для получения информации об обратной связи с пользователем.

Приемочные испытания: испытания системы программного обеспечения для подтверждения ее соответствия требованиям заказчика. Для приемочных испытаний сложных программно-аппаратных систем, для которых отсутствует предварительная информация об аналогичных системах, испытания на повышение надежности и стресс-тестирование [28] должны быть частью требований приемочных испытаний.

Регрессионные испытания: испытания программного обеспечения, которое было ранее протестировано, с целью выявления всех ошибок, внесенных при техническом обслуживании, разработке новых кодов, неправильной конфигурации или неадекватном контроле их причин.

c) Тестируемость программного обеспечения.

Тестируемость — это свойство программного обеспечения быть протестированным с минимальными затратами времени и ресурсов. Тестируемость является конструктивной характеристикой, кото-

рая позволяет результативно определить состояние программного обеспечения в эксплуатации. Характеристика тестируемости проекта также позволяет результативно выполнять процессы обнаружения неисправностей, их изоляции и диагностики. Для обеспечения тестируемости должна быть разработана структура функций программного обеспечения, для которых необходимо обеспечить возможность тестирования. Принцип модульного проектирования, при котором функция программного обеспечения не зависит от других функций, облегчает тестирование для обнаружения и устранения неисправностей. Такой подход делает удобным сопровождение функций программного обеспечения за счет упрощения процесса обновления или модификации программного обеспечения.

Программы самотестирования, процедуры мониторинга и контроля могут быть разработаны и включены в систему программного обеспечения для обеспечения самотестирования системы. Функции самотестирования могут быть выполнены по требованию или автоматически активироваться программами, используемыми для облегчения сопровождения, обслуживания и диагностики системы, и указывать ее состояние при эксплуатации. Особенности проекта самопроверки должны включать возможность обнаружения ложной тревоги при наличии ошибки оператора и состояния перехода. Ложная тревога — это предупреждение, сообщаемое функцией управления самодиагностикой, указывающее на наличие неисправности в работе, когда такая неисправность отсутствует. Количество сигналов ложной тревоги может быть уменьшено или доведено до нуля с помощью полного и точного диагностического анализа и валидированного процесса управления диагностикой при разработке системы.

Подход проектирования структуры для обеспечения тестируемости включает процесс разумного обоснования целей тестирования. Процесс анализирует параметры программного обеспечения и прогнозирует вероятность появления в программном обеспечении ошибок, которые могут быть обнаружены при тестировании. Анализ используют для оптимизации процесса тестирования и определения необходимого объема (охвата и глубины) тестирования. Анализ определяет необходимые средства управления ресурсами для тестирования и ценности или преимуществ конкретного подхода к тестированию.

d) Тестовые примеры.

Тестовые примеры разрабатывают на основе спецификаций программного обеспечения. Тестовые примеры используют для моделирования реальных условий эксплуатации программного обеспечения, связанных с исследуемыми аспектами или потенциальными проблемами программного обеспечения. Тестовый пример — это набор входных данных, условий выполнения и ожидаемых результатов, разработанных для конкретной цели тестирования; например, чтобы использовать определенный путь программного обеспечения или проверить соответствие определенным требованиям. Спецификация тестового примера — это документация, определяющая входные данные, ожидаемые результаты теста и условия работы тестируемого объекта. Результативный процесс тестирования предусматривает применение как ручных, так и автоматически созданных тестовых примеров. Ручные тесты охватывают всю глубину обнаружения неисправностей программного обеспечения и отражают понимание разработчиком проблемной области и структуры данных. Автоматические тесты охватывают широкий спектр исследования неисправностей, выполняя весь диапазон значений тестируемых параметров, в том числе крайние ситуации, которые могут пропустить ручные тесты. Процесс автоматического тестирования включает использование генератора тестовых примеров для принятия исходного кода, критериев тестирования, спецификаций или определений структуры данных в качестве входных данных для генерации тестовых данных и определения ожидаемых результатов. Тест на введение неисправности можно рассматривать в качестве одного из тестовых примеров, когда в одну из частей программного обеспечения вводят преднамеренную неисправность для проверки того, что другая часть реагирует соответствующим образом. Результаты тестирования используют для определения вероятных состояний неисправностей, это облегчает отказоустойчивое проектирование программного обеспечения. Метод введения неисправностей также используют для проверки охвата тестированием конкретной программой путем подсчета доли обнаруженных неисправностей.

Тестирование программного обеспечения — это попытка создать отказ программного обеспечения. Важно отметить, что для любого отказа необходимо формировать тестовый пример для включения его в набор тестов программного проекта. Наиболее важным аспектом стратегии тестирования является количество неисправностей, выявленных при тестировании, как функция времени. Это обеспечивает показатель эффективности теста.

e) Определение параметров программного обеспечения и параметры управления проектом.

Определение или оценка количественных значений параметров программного обеспечения необходимы для эффективного управления проектом. Оценки получают различными способами, описан-

ными ниже, для прогнозирования показателей безотказности программного обеспечения и повышения надежности работы системы.

- Показатели структуры проекта: показатели подхода к проектированию, сложности и независимости проекта программного обеспечения.
- Показатели полноты проектирования: показатель степени, в которой программное обеспечение полностью выполняет установленные функции.
- Показатели управления процессами: показатели управления, экономической эффективности и компромиссов при разработке программного обеспечения на основе результатов анализа данных процесса и соответствующих данных о неисправностях, собранных в процессе сбора данных.
- Показатели управления продуктом: характеристики программного обеспечения, установленные для программного продукта, разработанного на основе результатов анализа данных продукта и соответствующих данных о неисправностях, собранных в процессе сбора данных.

Многие из этих показателей используют в качестве исходных данных при разработке модели безотказности для прогнозирования и оценки количественных значений (при необходимости).

В приложении G представлен обзор параметров модели безотказности программного обеспечения, обычно используемых в отраслевой практике.

6.3.8 Повышение и прогнозирование безотказности программного обеспечения

Повышение безотказности программного обеспечения — это состояние, которое характеризуется повышением вероятности безотказной работы системы программного обеспечения во времени. Повышения безотказности программного обеспечения достигают за счет проектирования и прогрессивного достижения значения вероятности безотказной работы с помощью испытаний на повышение надежности. Программное обеспечение не отказывает, если только оно не предназначено для выявления отказов. Программа может отказать только после того, как она выполнена. Отказы программного обеспечения выявляют его неисправности, а устранение этих неисправностей приводит к повышению безотказности. Тенденции повышения безотказности программного обеспечения основаны на интенсивности устранения неисправностей по отношению к совокупному времени выполнения программного обеспечения. Для целей планирования время выполнения может быть преобразовано в календарное время для установления интенсивности отказов программного обеспечения и оценки вероятности безотказной работы. Программа повышения надежности [29] может быть установлена для комбинированной аппаратно-программной системы. Модели повышения надежности и методы оценки, основанные на данных об отказах, собранных в соответствии с программой повышения надежности, описаны в статистических методах повышения надежности [30]. Типичные методы улучшения проекта программного обеспечения приведены в 6.4. Испытания на повышение надежности программного обеспечения включают следующие действия.

a) Испытания на повышение надежности программного обеспечения.

Испытания на повышение надежности выполняют для оценки текущей вероятности безотказной работы, выявления и устранения неисправностей и прогнозирования будущей вероятности безотказной работы. Значения вероятности безотказной работы, основанные на подсчете количества неисправностей, обнаруженных и удаленных в процессе испытаний, сравнивают с промежуточными целевыми значениями вероятности безотказной работы программного обеспечения. Это способ выявления тенденций повышения безотказности в процессе тестирования для достижения целей надежности программного обеспечения.

Ускоренные испытания [31] успешно используют для сокращения времени испытаний. Для этого применяют повышенные нагрузки или увеличивают скорости применения повторяющихся нагрузок для оценки или демонстрации повышения надежности продукта. Для комбинированных программно-аппаратных систем применение к программному обеспечению повышенных нагрузок может охватывать различные сценарии эксплуатации. Цель состоит в верификации, адекватности работы системы в смоделированных условиях эксплуатации в течение времени испытаний.

b) Условия изготовления программного обеспечения.

Условия изготовления программного обеспечения включают аппаратную платформу, такую как аппаратная система, программное обеспечение операционной системы, параметры генерации системы, рабочую нагрузку и профиль эксплуатации. Профиль эксплуатации описан в 6.3.3.

Прогон — это результат выполнения программы. Прогон имеет идентифицируемые входные и выходные переменные. Испытания на надежность программного обеспечения основано на выборе набора значений входных переменных для конкретного прогона. Каждая входная переменная имеет объявленный тип данных, представляющий диапазон и порядок допустимых значений. Профиль экс-

плуатации — это функция, связанная с вероятностью входной переменной, которую используют в статистической оценке повышения надежности.

c) Несколько копий программного обеспечения.

Продолжительность тестирования во время испытаний на повышение надежности может быть сокращена за счет испытаний более одной копии программного обеспечения. Копии могут быть запущены одновременно для ускорения испытаний. Эта процедура позволяет выполнять прогон нескольких копий и суммировать время тестирования с повышенной скоростью процесса испытаний, это особенно важно для демонстрации достижения целей высокой надежности. В связи с этим общее количество календарного времени на тестирование сокращается. Использование нескольких копий может обеспечить преимущества, связанные с экономией средств и времени.

d) Прогнозирование безотказности программного обеспечения.

Повышение безотказности программного обеспечения — это улучшение безотказности программного обеспечения с течением времени, которое достигают путем систематического удаления из программного средства ошибок. Скорость повышения безотказности зависит от того, насколько быстро ошибки могут быть обнаружены и удалены. Модель безотказности программного обеспечения, применимая к условиям повышения безотказности, позволяет руководителям проекта отслеживать прогресс повышения безотказности программного обеспечения посредством статистических методов для установления тенденций и прогнозирования будущих целей в области безотказности. Если тенденция указывает на снижение безотказности, могут быть предприняты соответствующие меры управления.

e) Модели безотказности программного обеспечения.

Определение и прогнозирование показателей повышения безотказности программного обеспечения требует использования соответствующей модели безотказности программного обеспечения, которая описывает изменения безотказности программного обеспечения во времени. Параметры модели безотказности могут быть получены с помощью прогноза на основе экспериментальных данных или данных испытаний, полученных при тестировании системы. Выбор и использование модели безотказности программного обеспечения должны быть валидированы. Процесс оценки основан на моментах времени, когда происходит отказ, с достаточным объемом выборки для значимого накопленного времени работы. Это обеспечивает установление разумного уровня статистической достоверности для валидации тенденций повышения безотказности. Подход состоит в прогнозировании зрелости программного обеспечения и целей выпуска.

В приложении Н представлены примеры типичных моделей безотказности программного обеспечения, используемых в отраслевой практике.

6.3.9 Данные обратной связи о надежности программного обеспечения

Сбор данных о надежности программного обеспечения рассмотрен в 6.2. Действия по сбору данных проводят при эксплуатации, чтобы оценить надежность работы программного обеспечения в эксплуатации в помещениях заказчика. Это необходимо для подтверждения того, что принятый уровень надежности работы в условиях эксплуатации поддерживается, а также для совершенствования программного обеспечения. Информацию о надежности в эксплуатации собирают вместе с соответствующей информацией обратной связи с потребителем. Эту информацию используют для обоснования изменений в требованиях к новому программному обеспечению и начала разработки новой версии программного обеспечения.

Часто из-за динамики изменения условий применения и развития технологий на решение о выпуске нового программного обеспечения влияют рыночная конкуренция и бизнес-стратегии.

6.4 Повышение надежности программного обеспечения

6.4.1 Обзор повышения надежности программного обеспечения

Повышение надежности программного обеспечения может быть достигнуто с помощью улучшения проекта программного обеспечения, испытаний на повышение надежности и улучшения сопровождения и обслуживания программного обеспечения в службах поддержки потребителей, включая усилия по улучшению программного обеспечения.

Безотказность является ключевым свойством надежности программного обеспечения. Повышение безотказности программного обеспечения посредством испытаний на повышение надежности описано в 6.3.8. В следующих подразделах представлены практические подходы, относящиеся к проектированию безотказности программного обеспечения, и рекомендуемые методы улучшения и изготовления программного обеспечения. Целью проектирования является обеспечение тестируемости для просто-

ты верификации функций программного обеспечения, модульности для обеспечения независимости каждой программной функции и облегчения изоляции и локализации неисправностей, а также для простоты обслуживания модификации программного обеспечения на стадиях его жизненного цикла.

6.4.2 Снижение сложности программного обеспечения

a) Сложность структуры.

Сложность структуры описывает логические пути соединения программного обеспечения в проекте. Каждый модуль может быть запрограммирован (посредством кодирования) для обеспечения исполняемого модуля функции программного обеспечения в структуре программного обеспечения. Сложность структуры связана с тестируемостью программных кодов, которая влияет на обнаружение неисправностей, следовательно, влияет на безотказность и ремонтопригодность структуры программного обеспечения. Чем сложнее структура, тем сложнее протестировать программное обеспечение. Правила проектирования программного обеспечения должны устанавливать уровень сложности, чтобы облегчить обеспечение надежности проекта.

b) Функциональная сложность.

Функциональная сложность описывает требуемые функции, которые должен выполнять программный модуль или сегмент кода в модуле. В идеале для выполнения одной функции должен быть спроектирован один модульный блок с одним набором соответствующих входов и выходов, чтобы облегчить изоляцию и устранение неисправностей программного обеспечения. На практике как сложность структуры, так и функциональную сложность необходимо учитывать при оценке проекта программного обеспечения. Стратегия проектирования программного обеспечения по сложности напрямую связана с количеством тестовых примеров, необходимых для полной верификации программного обеспечения.

6.4.3 Устойчивость программного обеспечения к неисправностям

Устойчивость программного обеспечения к неисправностям — это способность программного обеспечения продолжать работу и сохранять целостность данных при наличии определенных неисправностей. Устойчивость программного обеспечения к неисправностям необходима для предотвращения неисправностей, вызывающих отказ системы во время работы системы. Устойчивость программного обеспечения к неисправностям конструируют так, чтобы иметь низкую вероятность проявления отказов общего вида на основе анализа различных конструкций систем, включая следующие рекомендуемые методы.

- Ограничение неисправности: программное обеспечение должно быть написано так, чтобы при возникновении неисправности она не могла распространяться на части программного обеспечения за пределами локального домена, где она произошла.
- Обнаружение неисправности: программное обеспечение должно быть написано так, чтобы оно проверяло и реагировало на неисправности при их возникновении.
- Устранение неисправности: программное обеспечение должно быть написано так, чтобы после обнаружения неисправности могли быть выполнены действия, обеспечивающие продолжение успешного функционирования программного обеспечения.
- Разнообразие проектов: программное обеспечение и его данные создают таким образом, чтобы были доступны резервные версии.

Устойчивость программного обеспечения к неисправностям проявляет свойство постепенной деградации системы, которое позволяет ей продолжать работать должным образом в течение некоторого времени в случае отказов. Это предотвращает отказы, которые в противном случае могут привести к внезапному отключению системы или полному ее разрушению. Устойчивость программного обеспечения к неисправностям имеет особое значение для систем, критичных по отношению к безопасности, которые зависят от высокой готовности работы системы в случае неисправностей или работы в неблагоприятных условиях. Примером проекта, устойчивого к неисправностям, является протокол управления передачей для интернет-связи. Это программный протокол, разработанный для обеспечения безотказной двусторонней связи в сети с коммутацией пакетов, даже при наличии несовершенных или перегруженных линий связи. Устойчивости проекта к неисправностям достигают за счет требования того, чтобы в конечных точках связи целостность данных не нарушалась, а лишь снижалась пропускная способность на пропорциональную величину при продолжении работы.

Метод многовариантного программирования является возможным подходом, используемым для обеспечения отказоустойчивости при проектировании критических систем и для повышения безотказности работы программного обеспечения. Этот метод задействует несколько функционально эквивалентных программ, которые независимо генерируются на основе одних и тех же начальных спецификаций программного обеспечения. Независимость отдельных усилий программирования значительно умень-

шает вероятность идентичных неисправностей программного обеспечения, возникающих в двух или более версиях программы. Реализация этих программ использует разные алгоритмы и язык программирования. В программное обеспечение встраивают специальные механизмы, позволяющие управлять этими отдельными программами посредством схемы голосования в алгоритме принятия решения для выполнения программы. Концепция основана на предположении, что выходные данные нескольких независимых версий с большей вероятностью будут правильными, чем выходные данные одной версии с точки зрения резервирования. На практике преимущества улучшения многопрограммных усилий требует обоснования дополнительных затрат времени и ресурсов для обеспечения их рентабельной реализации. Результативность метода также зависит от обеспечения различных характеристик неисправностей в рамках версии при разработке и реализации программного обеспечения.

6.4.4 Функциональная совместимость программного обеспечения

Функциональная совместимость программного обеспечения — это способность различных систем программного обеспечения работать вместе для обмена информацией и ее использования. В открытой системе, такой как IP-сеть, важно обеспечить взаимодействие различных систем программного обеспечения для установления линий связи. Отказ линии связи может повлиять на надежность работы системы. Один из практических подходов, рекомендуемых для улучшения взаимодействия в сети связи, заключается во включении специального средства в проект системы программного обеспечения для мониторинга ситуации в установленной связи. Например, технология «сердцебиение» (схема обработки и синхронизации сигналов), включающая функцию мониторинга, заключается в отправке сигнала «сердцебиения» друг другу, когда установлена линия связи. Если связь прервана из-за изменений в условиях работы или по каким-либо другим причинам, система программного обеспечения автоматически пытается найти и восстановить соединение, чтобы поддерживать непрерывную связь, так чтобы надежность работы сети связи не ухудшалась.

6.4.5 Повторное использование программного обеспечения

Повторное использование программного обеспечения обусловлено различными причинами, в том числе сведениями об эксплуатации, экономией времени и средств, а также данными о запатентованных продуктах при принятии деловых решений.

Повторное использование программного обеспечения — это использование существующего программного обеспечения для создания нового программного обеспечения. Повторно используемое программное обеспечение является повторно используемым активом. Наиболее известным повторно используемым активом является код. Программный код, написанный один раз, можно использовать в другой программе, написанной позднее. Повторное использование программного кода является распространенным приемом, который позволяет сэкономить время и усилия за счет уменьшения количества необходимых операций. Повторное использование программного обеспечения — это степень, в которой программный актив может быть использован в другой системе программного обеспечения или при создании других активов.

С точки зрения надежности, применение программного обеспечения для повторного использования в проектах и возможность его повторного использования необходимо контролировать для повышения надежности. Возможность многократного использования напрямую зависит от структуры программного обеспечения и его модульного проекта. Для того, чтобы программный блок можно было многократно использовать, он должен ограничиваться полным выполнением только одной функции. Это ограничение является существенным, потому что, если предполагаемое повторное использование программного модуля выполняет менее одной функции, или если оно способно выполнять более одной функции, его трудно реализовать или поддерживать для предполагаемого нового назначения. Отклонение от такого ограничения уменьшает полезность программного обеспечения для многократного использования. Повторное использование программного обеспечения, которое не выполняет только одну функцию, может отрицательно влиять на надежность из-за возможных ошибок, введенных в программное обеспечение во время изготовления или обслуживания.

Повторное использование программного обеспечения необходимо применять только в том случае, если функциональные требования нового программного блока соответствуют требованиям к программному обеспечению многократного использования для очень похожих областей применения и условий эксплуатации. В противном случае это снизит экономическую эффективность повторного использования программного обеспечения и, возможно, безотказность при выполнении.

Программное обеспечение для многократного использования должно быть хорошо документировано для отслеживания и облегчения управления конфигурацией программных активов. Покупные готовые (COTS) программные средства и системы следует рассматривать как программное обеспе-

чение многократного использования для различных применений. В целях обеспечения достоверности следует проводить квалификационные испытания для валидации работы и соответствия программного продукта/системы требованиям проекта.

6.4.6 Обслуживание и улучшение программного обеспечения

Обслуживание программного обеспечения — это модификация программного продукта после поставки для устранения неисправностей, улучшения работы или других свойств программного обеспечения, а также для адаптации продукта к измененным условиям работы. Существует четыре основные категории обслуживания программного обеспечения.

- Корректирующее обслуживание: модификация программного продукта, выполняемая после поставки для устранения обнаруженных проблем.
- Адаптивное обслуживание: модификация программного продукта, выполняемая после поставки, для обеспечения возможности использования программного продукта в измененных или изменяющихся условиях.
- Безупречное обслуживание: модификация программного продукта после поставки для улучшения его работы или обслуживания.
- Профилактическое (превентивное) обслуживание: модификация программного продукта после поставки для обнаружения и исправления ошибок в программном продукте, прежде их дальнейшего распространения до появления отказов.

Ключевыми вопросами обслуживания программного обеспечения являются управление и технические аспекты. Вопросы управления включают согласование с приоритетами потребителей, планирование и распределение ресурсов для технического обслуживания, обучение обслуживающего персонала, работы по техническому обслуживанию в соответствии с контрактом и отзывы об удовлетворенности пользователей. Технические проблемы включают в себя записи об инцидентах, решение технических задач, анализ влияния, стандартизацию процедур применения и методов тестирования, оценку ремонтопригодности программного обеспечения и оценку эффективности тестирования.

Улучшение программного обеспечения является частью процесса эволюции программного обеспечения. Система программного обеспечения в условиях эксплуатации отличается более высокой сложностью, связанной с работой по модификации и совершенствованию, выполняемой для удовлетворения потребностей потребителей, постоянными изменениями в стратегиях поддержки обслуживания благодаря конкурентоспособным предложениям услуг и необходимостью развивать навыки и методы адаптации к изменяющейся бизнес-среде. Степень и успех усилий по сопровождению и улучшению программного обеспечения должны быть верифицированы, валидированы и документированы. Установленные ресурсы, необходимые для обслуживания программного обеспечения, должны быть частью стратегии обеспечения надежности.

6.4.7 Документация программного обеспечения

Документация программного обеспечения — это текст и информация о документации, проекте и его применение. Разработка документации является важной частью разработки программного обеспечения. Основные категории документации программного обеспечения включают следующее.

а) Документация по проекту и его структуре.

В документации представлено описание программного продукта, условий его применения и принципов построения, которые должны быть использованы при проектировании. Документ программного обеспечения содержит всестороннюю информацию о модели проекта программного обеспечения со всеми деталями.

- Проект данных содержит описание структуры программного обеспечения. Описание объектов данных и взаимосвязей между ними определяет выбор структур данных, которые влияют на сложность структуры программного обеспечения и на надежность его работы.
- Проект структуры, используя характеристики информационного потока, формирует структуру программы, что влияет на модульность проекта программного обеспечения и его безотказность. Должны быть использованы рекомендуемые методы описания структуры [32].
- Проект интерфейса описывает внутренние и внешние интерфейсы программы, а также проект интерфейсов пользователя, включая аппаратные интерфейсы и драйверы аппаратных средств. Проект внутренних и внешних интерфейсов основан на информации, полученной в результате анализа проекта, которая влияет на схемы резервирования и требования к безотказности систем, программных и аппаратных модулей и конфигураций.
- Проект процедур описывает концепции структурированного программирования с использованием графических, табличных и текстовых обозначений. Эти средства разработки позволяют разработ-

чику представлять детали процедур, которые облегчают кодирование, влияют на последовательность методов программирования и уменьшают количество ошибок при кодировании.

b) Техническая документация.

Техническая документация включает коды, алгоритмы, интерфейсы и дополнительное описание различных аспектов предполагаемой работы программных продуктов. Документация должна быть полной, но сжатой при написании исходного кода, чтобы облегчить обслуживание и обновление программного обеспечения. Комментарии в коде — это форма технической документации, когда комментирование включает краткие пояснения, которые распознаются компилятором только как комментарии, а не как часть кода программ. Целью использования комментариев в коде программного обеспечения является повышение возможности повторного использования и обслуживания. Это облегчает анализ и контроль кодов, их обновление и модификацию, а также повышает целостность и безотказность программного обеспечения. Техническая документация может также включать, где это необходимо, спецификации требований, планы и процедуры испытаний, технические отчеты и соответствующие данные.

c) Документация пользователя.

Документация пользователя [33] включает в себя руководства для конечных пользователей, системных администраторов и вспомогательного персонала. Она предназначена для оказания помощи конечному пользователю в применении программных продуктов. Документация содержит описание использования программного обеспечения в условиях его применения. Документация пользователя также включает описание функций программного средства и помогает пользователю реализовать эти функции при использовании; она включает информацию о выпуске программного обеспечения и управлении версиями, инструкции по устранению неисправностей, инструкции по безопасности, предупреждения и ограничения по применению программного обеспечения, а также справочные инструкции. Для обеспечения удовлетворенности потребителей и обеспечения удобного доступа и обслуживания возможна онлайн-помощь.

d) Маркетинговая документация.

Маркетинговая документация включает в себя рекламные материалы, побуждающие случайных наблюдателей узнать больше о программном средстве. Центры доступа в интернет и обслуживания потребителей являются распространенными услугами в современной конкурентной рыночной среде для получения программных продуктов и информации об их применении. Ориентация на потребителя имеет важное значение в разработке маркетинговой стратегии. Маркетинговая документация является лишь одним из многих подходов к распространению информации. Маркетинговая документация может включать информацию, касающуюся определенных значений показателей надежности и связанных с этим вопросов для соответствующих применений программного обеспечения, таких как повторное использование и модификация программного обеспечения для конкретных областей применения.

6.4.8 Автоматизированные средства

Автоматизированные средства полезны для обычной обработки данных, анализа вычислений и сравнения результатов оценки. На рынке имеется широкий спектр автоматизированных средств, которые удовлетворяют большей части потребностей применения при моделировании, анализе и ведении базы данных для поддержки всех форм оценки надежности. Выбор и применение соответствующих средств для конкретных задач проекта требует от инженеров или специалистов в области надежности знаний и опыта. Автоматизированные средства позволяют системам, которые могут помочь рутинной вычислительной работе, повысить производительность. Достоверность и точность этих автоматизированных средств должны быть исследованы до принятия решения об их применении в проекте. Возможность поддержки этих средств должна быть определена до их приобретения. Автоматизированные средства, используемые для разработки и тестирования (испытаний) программного обеспечения, применимы для испытаний на повышение безотказности. Они составляют важную часть при верификации программного обеспечения и валидации системы программного обеспечения.

6.4.9 Техническая поддержка и обучение пользователей

Техническая поддержка — это комплекс услуг по оказанию помощи в использовании программных средств. Целью является помощь пользователю в решении конкретных вопросов работы или применения программного средства. Техническая поддержка принимает различные формы, включая запрос по телефону, онлайн-услугу, запрос по электронной почте, восстановление удаленного доступа и выезд специалиста на место. Организации, занимающиеся разработкой технологических продуктов, все чаще расширяют и используют независимые внешние центры обработки вызовов по деловым, экономическим и географическим причинам, чтобы обеспечить оперативное реагирование на запрос услуги технической поддержки. Эти центры обработки вызовов обеспечивают централизованную техническую

поддержку для широкого спектра технологических продуктов, таких как компьютерные системы, включая программное обеспечение, требующее круглосуточного технического сопровождения с бесплатным доступом пользователей по всему миру. Услуги технической поддержки являются частью сопровождения программного средства для обеспечения работоспособности и безотказности продукта, а также повышения его надежности.

Обучение пользователей программного обеспечения является важным аспектом повышения его надежности. Цель состоит в повышении понимания пользователями применения программного обеспечения и повышения уровня навыков в работе с ним. Обучение пользователей осуществляют в различных формах, включая онлайн-доступ к учебной базе данных поставщика, помощь колл-центра, специализированные услуги технического эксперта для решения нетипичных проблем.

7 Гарантия на программное обеспечение

7.1 Общие понятия гарантии на программное обеспечение

Гарантия на программное обеспечение — это запланированный и систематизированный набор действий, которые обеспечивают соответствие процессов жизненного цикла программного обеспечения и его самого требованиям, стандартам и процедурам. Модели зрелости возможностей [8, 9] являются распространенным методом управления, рекомендуемым для реализации программ гарантии на программное обеспечение в организациях, разрабатывающих программное обеспечение. Существуют также хорошо документированные методологии и процедуры формирования гарантии на программное обеспечение, применяемые при его разработке и применении [34].

Формирование гарантии на программное обеспечение обычно включает применение технических дисциплин в области качества, надежности, безопасности и защищенности, связанных с разработкой программного продукта и работой системы. Гарантия на программное обеспечение является основой доверия и принятия решений при планировании, разработке, обслуживании программного обеспечения. Гарантией управляют в соответствии с жизненным циклом [35] для проверки соответствия программных продуктов установленным целям для достижения соответствующей безопасности, защищенности, надежности и других целей. Исследования на основе гарантийного случая [36] представляют собой записи о претензиях к работе процесса, а также к физическим свойствам и функциональным характеристикам системы программного обеспечения, проверенным для подтверждения соответствия спецификациям системы. Гарантию на программное обеспечение учитывают при оценке рисков, верификации и валидации тестирования (испытаний), документировании и ведении записей аудита в качестве объективных свидетельств. При формировании гарантии используют данные соответствующих оценок на основе проекта для мониторинга программного продукта и соответствующего процесса возможного улучшения.

В надежности программного обеспечения главную роль играет безотказность программного обеспечения, на основе которой формируют гарантию на программное обеспечение посредством реализации процесса разработки программного обеспечения и обеспечения его безотказности [37]. Надежность и качество программного обеспечения являются необходимыми условиями обеспечения безопасности и защищенности при работе системы.

7.2 Процесс адаптации

Технологический процесс — это деятельность по управлению проектом для обеспечения выполнения сроков и необходимых действий, а также по соответствующему распределению ресурсов для удовлетворения потребностей проекта. Процесс адаптации может быть использован для выполнения гарантийных обязательств по программному обеспечению. Процесс адаптации часто используют в краткосрочных проектах для улучшения или поддержания работы системы, когда требования и ограничения проекта являются более строгими, чем в начале разработки нового проекта. Для реализации процесса адаптации проекта рекомендуется выполнение следующих задач.

- Выявление и документирование обстоятельств, влияющих на адаптацию, таких как условия эксплуатации, объем и сложность проекта, график и бюджет проекта, доступность ресурсов, проблемы безопасности, защищенности и целостности проблемы предыдущих моделей и требования соответствия стандартам.
- Определение входных требований для принятия решений.
- Установление целей проекта и плана процесса адаптации.

- Выбор подходящих стадий жизненного цикла, применимых для адаптации с целью достижения намеченных результатов.

- Документирование результатов адаптации для облегчения анализа его результативности и улучшения проекта.

7.3 Влияние технологий на гарантию на программное обеспечение

Существует большое количество усовершенствований программных технологий для эффективной разработки программного обеспечения, что привело к универсальности и экономическим преимуществам их применения. Гарантия на программное обеспечение традиционно фокусируется на обеспечении качества и безотказности программного обеспечения при его разработке. Кибератаки на программное обеспечение стали более частыми, заметными и все более изощренными. Они влияют не только на разработку программного обеспечения с применением COTS, но и вызывают проблемы, связанные со значительной потерей времени у операторов и пользователей систем программного обеспечения. Вся ситуация превратилась в цепочку реакций, распространяемых неизвестными вирусами, скрытыми вторжениями и кибератаками, создавая сложную динамичную среду риска для ИТ-действий, которые зависят от программного обеспечения.

Гарантия на программное обеспечение имеет решающее значение для организаций, занимающихся безопасностью и финансовыми транзакциями, ввиду уязвимости применения программных средств. Гарантия на программное обеспечение охватывает разработку и выполнение методов и процессов обеспечения функционирования программного обеспечения должным образом, с одновременным снижением риска уязвимостей, вредоносных кодов, неисправностей и ошибок, которые могут причинить вред конечному пользователю. Гарантия программного обеспечения жизненно важна для обеспечения безопасности критически важных ИТ-ресурсов. В связи с быстро меняющимися свойствами опасных условий, даже программное обеспечение самого высокого уровня не защищено от кибер-вторжений, если программное обеспечение имеет неправильную конфигурацию и неправильное обслуживание. Управление угрозами в киберпространстве требует многоуровневого подхода к предотвращению нарушения безопасности и совместной работе. Разработчики создают более безопасное и устойчивое программное обеспечение, интеграторы системы гарантируют, что программное обеспечение установлено правильно, операторы правильно выполняют поддержку системы, а конечные пользователи используют программное обеспечение безопасным и надежным образом.

Это приводит к тому, что организации, связанные с программным обеспечением, переопределяют гарантийные обязательства на программное обеспечение при его эксплуатации. Например, гарантию на программное обеспечение можно интерпретировать как «уровень уверенности в том, что программное обеспечение не содержит уязвимостей, намеренно или непреднамеренно встроенных в программное обеспечение, либо случайно введенных в любое время в течение жизненного цикла программного обеспечения, и что программное обеспечение функционирует по назначению» [38]. Гарантия на программное обеспечение должна обеспечивать разумный уровень обоснованной уверенности в том, что программное обеспечение функционирует правильно и предсказуемо в соответствии с документированными требованиями. Целью гарантии на программное средство является обеспечение того, что функционирование программного обеспечения не может быть скомпрометировано ни путем прямой атаки, ни путем воздействия со стороны вредоносного кода.

В зависимости от конкретного применения уровень доверия к гарантии на программное обеспечение связан:

a) с достоверностью — отсутствием уязвимостей, которые могут быть использованы злонамеренно или непреднамеренно;

b) предсказуемым выполнением — программные функции при выполнении по назначению обеспечивают обоснованное доверие;

c) соответствием — запланированный и систематический набор междисциплинарных действий для гарантии того, что программные процессы и продукты соответствуют требованиям, стандартам и процедурам.

Проблемы, определенные для обеспечения гарантии на программное обеспечение, включают в себя:

1) случайные ошибки проектирования или изготовления, которые приводят к уязвимостям кода;

2) изменяющуюся технологическую среду, которая выявляет новые уязвимости и обеспечивает кибер-преступников новыми методами работы;

3) злонамеренных инсайдеров и посторонних, которые стремятся причинить вред разработчикам или конечным пользователям.

Первая проблема — случайная и непреднамеренная. Вторая и третья проблемы являются преднамеренными и задуманными для кибератак. Контрмеры заключаются в управлении рисками, связанными с этими проблемами, с помощью передовых методов обеспечения гарантии на программное обеспечение.

7.4 Лучшие практики гарантии на программное обеспечение

Существуют форумы по программным технологиям и гарантии на программное обеспечение [39], в которых участвуют правительства, промышленность, научные круги и пользователи, участвующие в реализации передовых методов гарантии на программное обеспечение. Рекомендуемые методы разработки программного обеспечения указаны в 6.1. Рекомендуемые лучшие практики гарантии на программное обеспечение представлены ниже:

a) установление политики гарантии на программное обеспечение для управления разработкой программного обеспечения и процессом его изготовления;

b) обучение применению методов, связанных с программным продуктом, и использованию справочных ресурсов;

c) использование общей платформы проектирования структуры программного обеспечения для содействия разработке разнообразных программных продуктов;

d) выполнение процессов жизненного цикла программного обеспечения;

e) инициирование исследований гарантийных случаев программного обеспечения для оценки рисков, где это оправдано и целесообразно;

f) установление общих критериев для верификации и валидации квалификации и соответствия программного обеспечения;

g) управление конфигурацией для выпуска версий программного обеспечения;

h) установление системы отслеживания работы и неисправностей программного обеспечения и системы сбора данных для разработки и улучшения процессов программного обеспечения;

i) создание центра поддержки потребителей для облегчения обслуживания пользователей и применения программного продукта.

Приложение А
(справочное)

Категоризация и применение программного обеспечения

А.1 Категоризация программного обеспечения

А.1.1 Категории программного обеспечения

Категории программного обеспечения включают результаты разработки программного обеспечения и данные, созданные в процессе разработки программного обеспечения. Характеристики и условия применения программного обеспечения являются факторами, влияющими на надежность при разработке и изготовлении программного обеспечения. Схема категоризации представляет собой упорядоченную комбинацию представлений и категорий, связанных с программным обеспечением [40].

Категория представляет собой группировку программного обеспечения на основе его свойств и характеристик. Для облегчения разработки и применения программного обеспечения упор делают на его надежность. Типичные примеры группировки программного обеспечения описаны ниже.

А.1.2 Характеристики

- Режим работы: категории, определяемые установленным методом или типом обработки данных, принятым системой программного обеспечения (обработка в режиме реального времени, пакетная обработка, обработка с разделением по времени, параллельная и согласованная обработка). Система, работающая в режиме реального времени, должна быть сфокусирована на времени отклика. Программное обеспечение с разделением времени должно быть сфокусировано на спецификациях интерфейса.

- Масштаб программного обеспечения: категории, определяемые объемом (например, KSLOC) или сложностью (например, потоком данных) программного обеспечения и интерпретируемые как небольшое, среднее или большое программное обеспечение; программное обеспечения простое или сложное программное обеспечение. Сложное или большое программное обеспечение должно быть разделено на более мелкие части, чтобы облегчить управление проектом, поэтапное тестирование и интеграцию.

- Стабильность: категории, определяемые внутренним эволюционным аспектом или стабильностью с точки зрения характеристик системы программного обеспечения, таких как постоянно меняющиеся, постепенно увеличивающиеся или маловероятные изменения. В случае постоянных или постепенно увеличивающихся изменений программного обеспечения необходимы спецификации интерфейса, которые обеспечивают гибкость и стабильность после каждого изменения. Часто используют специальные модели разработки, такие как спиральная модель и модель водопада.

- Функция программного обеспечения: категории, определяемые типом функции, таким как компилятор, обработка бизнес-транзакций, обработка текстов, система управления. Для бизнес-транзакций должны быть обеспечены безопасность и доступность. Системы управления должны быть сосредоточены на доступности, безопасности и защищенности.

- Безопасность: категории, определяемые уровнем защиты от несанкционированного доступа, наличием контрольного журнала, защитой программ и данных. Акцент должен быть сделан на устойчивость (робастность) и готовность.

- Безотказность: категории, определяемые уровнем требуемой безотказности, таким как зрелость, устойчивость к неисправностям и восстанавливаемость. Акцент должен быть сделан на повышении безотказности и управлении конфигурациями для обеспечения безотказности.

- Работа: категории, определяемые работой программного обеспечения с точки зрения мощности, пропускной способности и времени отклика. Акцент должен быть сделан на время отклика, которое зависит от нагрузки и мощности.

- Язык: категории, определяемые типом языка программирования, в основном использованном в программном обеспечении, таком как традиционный (например, COBOL, FORTRAN), процедурный (например, C), функциональный (например, Lisp), объектно-ориентированный (например, C++). Особое внимание следует уделять обучению программистов и знакомству пользователей с особенностями и ограничениями языка программирования.

А.1.3 Условия окружающей среды

- Область применения: категории, определяемые типом или классом внешней системы, в которой используют программное обеспечение, например, электронный бизнес, управление процессами и сетевая система. Безопасность и целостность данных являются основными факторами, влияющими на электронный бизнес. Безопасность и безотказность — важные проблемы в управлении процессом. Время отклика и готовность имеют решающее значение для работы сетевых систем.

- Компьютерная система: категории, определяемые конкретной целью компьютерной системы, в которой работает программное обеспечение, например, управляемая микропроцессором система, мэйнфрейм (центральный блок обработки данных) и операционная система в режиме реального времени. Ограничения объема памяти и программного кода важны для микропроцессорных систем. Следует учитывать совместимость программного

обеспечения в конфигурации аппаратного обеспечения мэйнфрейм и времени отклика для работы в режиме реального времени.

- Класс пользователя: категории, определяемые уровнем квалификации или характеристиками предполагаемого класса пользователя, такого как новичок, средний или эксперт. Идентификация класса пользователей имеет важное значение при проектировании интерфейса и разработке инструкций пользователя для облегчения применения.

- Компьютерный ресурс: категории, определяемые ограничениями компьютерного ресурса, такими как требования к памяти, требования к диску и требования к локальной сети. Ограничения компьютерного ресурса влияют на развитие потребностей в сопровождении программного обеспечения, а также возможностей его применения.

- Критичность программного обеспечения: категории, определяемые требованиями к уровню целостности продукта, такими как требования национальной безопасности, безопасности организации и конфиденциальности. Нормативные требования и социальные потребности должны быть приняты во внимание.

- Доступность программного продукта: категории, определяемые доступностью программного продукта, такие как коммерческое приобретение (COTS), пользовательское или фирменное программное обеспечение. Сроки приобретения и доступность программного продукта определяют решение о собственном проектировании или аутсорсинге в управлении проектом.

А.1.4 Данные

- Представление данных: категории, определяемые объектом данных, их типом и структурой, такими как реляционный, индексированный, отформатированный файл. Должна быть учтена совместимость данных.

- Использование данных программного обеспечения: категории, определяемые типом использования предполагаемых данных программного обеспечения, таким как единственный пользователь, несколько последовательных пользователей. Использование данных может повлиять на проект файла данных и критерии поддержки при обслуживании данных.

А.2 Применения программного обеспечения

Программное обеспечение используют для различных применений. Обычно применения программного обеспечения могут быть сгруппированы следующим образом.

- Программное обеспечение системы: программное обеспечение, обеспечивающее инфраструктуру для управления аппаратным обеспечением компьютера, чтобы применение программного обеспечения могло быть выполнено. Примерами являются операционные системы, такие как системы Microsoft Windows, Mac OS и Linux.

- Прикладное программное обеспечение: компьютерное программное обеспечение, предназначенное для облегчения пользователю выполнения определенных задач, таких как обработка текста, работа с электронными таблицами и использование баз данных.

- Встроенное программное обеспечение: программное обеспечение, хранящееся в программируемых устройствах памяти продуктов конечного пользователя для внутреннего контроля различных электронных устройств, таких как пульты дистанционного управления, калькуляторы, мобильные телефоны и цифровые камеры.

- Промежуточное программное обеспечение: компьютерное программное обеспечение, которое соединяет элементы программного обеспечения для большого количества приложений или предоставления таких услуг, как многопроцессорная обработка в распределенных системах и веб-службах.

- Тестовое программное обеспечение: подмножество программного обеспечения, специально предназначенное для тестирования программного обеспечения и автоматизации тестирования.

- Программное обеспечение для программирования: инструмент разработки программного обеспечения, который помогает разработчикам программного обеспечения создавать, отлаживать, поддерживать и сопровождать другие программы и приложения. Примеры включают инструменты CASE.

- Вредоносное программное обеспечение: программное обеспечение, предназначенное для проникновения в компьютер без согласия владельца.

Приложение В
(справочное)

Требования системы к программному обеспечению и соответствующие действия по обеспечению надежности

В.1 Общие положения

Обычно требования системы к программному обеспечению и соответствующие действия по обеспечению надежности на каждой стадии жизненного цикла программного обеспечения объединяют. Эта информация может быть использована в качестве базовой для адаптации проектов обеспечения надежности программного обеспечения.

В.2 Определение требований

Требования системы к программному обеспечению	Соответствующие действия по обеспечению надежности
- Рыночная информация о программных продуктах - Требования к применению системы и потребности пользователей - Домен и платформа операционной системы	- Идентификация требований к программному обеспечению - Идентификация потребности в работе системы - Идентификация потребности в поддержке (сопровождении) системы

В.3 Анализ требований

Требования системы к программному обеспечению	Соответствующие действия по обеспечению надежности
- Требования к функциям и возможностям работы - Сценарии применения - Установленные требования к безопасности, защищенности и целостности применения, где это применимо - Требования к интерфейсу - Квалификационные требования - Возможности проекта и тестируемость программного обеспечения - Возможность эксплуатации и обслуживания - Требования к установке и приемке - Требования к документации	- Разработка профиля эксплуатации - Разработка плана проекта надежности - Разработка плана обеспечения надежности - Идентификация показателей надежности программного обеспечения - Определение требований к целостности данных - Определение требований безопасности и защищенности - Установление правил разработки проекта с учетом человеческого фактора (эргономики) - Установление критериев поддержки программного обеспечения - Идентификация ограничений, влияющих на проектирование и реализацию надежности, включая специфические требования для включения в проект - Установление критериев повторного использования программного обеспечения - Установление критериев повышения безотказности программного обеспечения и приемлемости квалификации - Определение записей об испытаниях на надежность и требований к документации

В.4 Структурное проектирование и разработка

Требования системы к программному обеспечению	Соответствующие действия по обеспечению надежности
- Структура, описывающая верхний уровень и определяющая составляющие элементы программного обеспечения - Преобразование и распределение требований для облегчения компоновки объектов программного обеспечения - Включение специфических требований безопасности, защищенности и целостности, где это необходимо, в структуре системы - Внутренние и внешние интерфейсы для интеграции и проверки системы - Предварительная документация по базе данных и требованиям к испытаниям - Рекомендуемые методы проектирования и стандарты для выполнения целей проекта и проектных спецификаций - Прослеживаемость требований программного объекта - Возможность детального проектирования - Условия эксплуатации и обслуживания	- Выполнение анализа сценария применения - Определение структурной и функциональной сложности программного обеспечения - Включение специфичных для применения требований при моделировании надежности системы - Выполнение анализа функциональной модели готовности/безотказности - Выполнение распределения готовности/безотказности по блокам программного обеспечения - Создание базы данных показателей надежности - Выполнение предварительного прогноза готовности/безотказности - Разработка плана повышения надежности программного обеспечения и плана принятия квалификации - Установление системы регистрации данных и отчетности - Установление плана поддержки программного обеспечения - Анализ структуры проекта для его реализации

В.5 Детальное проектирование

Требования системы к программному обеспечению	Соответствующие действия по обеспечению надежности
- Уточненный нижний уровень структуры для кодирования программных модулей и включения в объекты конфигурации программного обеспечения - Подробные технические характеристики программных модулей и описания объектов конфигурации программного обеспечения - Согласованность и прослеживаемость подробных спецификаций проекта и структуры - Установление методов проектирования и стандартов для удовлетворения требований проекта - Установление специальных методов проектирования для решения вопросов безопасности, защищенности и целостности, где это применимо - Все требования к интерфейсу для компиляции и тестирования программных модулей и объектов конфигурации - Документирование требований к базе данных и к тестированию (испытаниям) и графики тестирования - Анализ управления проектом для мониторинга прогресса и достижения поставленных целей - Базовый уровень конфигурации программного обеспечения и обмен информацией об изменениях проекта	- Внедрение правил проектирования программного обеспечения - Установление стандартов и критериев оценки показателей - Введение специальных проектов для удовлетворения требований безопасности, защищенности и целостности - Повышение отказоустойчивости проекта - Применение стандартов надежности программного обеспечения - Проведение анализа и контроля кода программного обеспечения - Уточнение распределения готовности/безотказности программного обеспечения - Выполнение оценки сложности программного обеспечения - Прогнозирование безотказности модулей программного обеспечения - Прогнозирование показателей готовности/безотказности подсистем программного обеспечения - Выполнение анализа компромиссов проекта - Уточнение прогноза показателей готовности/безотказности программного обеспечения - Обновление базы данных показателей надежности - Выполнение управления конфигурацией - Выполнение документального анализа проекта - Выполнение анализа проекта

В.6 Изготовление

Требования системы к программному обеспечению	Соответствующие действия по обеспечению надежности
- Методы и стандарты проектирования модулей и кодирования программного обеспечения и стандарты - Объекты конфигурации программного обеспечения с определенными программными модулями - Критерии верификации для тестирования модулей - Охват тестированием модулей программного обеспечения - Верификация функций программного обеспечения, включая применение спецификаций на требования безопасности, защищенности и целостности - Возможность интеграции и тестирования программного обеспечения	- Выполнение стандартов и критериев оценки показателей - Определение охвата кодированием модулей программного обеспечения - Выполнение тестирования модулей - Определение охвата неисправностей и завершение тестирования - Классификация данных о неисправностях - Выполнение процесса гарантии надежности для тестирования модулей и функций - Верификация модулей и функций на соответствие спецификаций на работу и применение программного обеспечения - Установление отчетности, анализа и корректирующих действий в случае отказа - Выполнение программы гарантии на программное обеспечение, включая аутсорсинг и цепочку поставок, где это необходимо - Выполнение анализа проекта

В.7 Интеграция

Требования системы к программному обеспечению	Соответствующие действия по обеспечению надежности
- Стратегия интеграции для модулей и конфигурации программного обеспечения - Критерии верификации для тестирования объекта конфигурации программного обеспечения - Верификация подсистем программного обеспечения, включая спецификации на применение и требования безопасности, защищенности и целостности - Документирование результатов тестирования интеграции - Документация об изменениях проекта - Стратегия регрессии для повторной верификации измененных объектов - Система сбора данных тестирования (испытаний)	- Выполнение процедуры отслеживания неисправностей - Выполнение процедуры анализа неисправностей - Инициирование программы повышения надежности - Выполнение записей об отказах, их анализа и корректирующих действий в системе - Выполнение системы сбора данных - Проверка подсистем программного обеспечения для интеграции - Выполнение тестирования интеграции - Определение оценок показателей готовности/безотказности по данным испытаний - Определение проблемных областей - Выполнение корректирующих действий - Управление изменениями проекта и выпусками версий - Проведение документального анализа проекта

В.8 Приемка

Требования системы к программному обеспечению	Соответствующие действия по обеспечению надежности
- Критерии приемки системы программного обеспечения - Демонстрация соответствия результатов испытаний (тестирования) установленным требованиям - Валидация результатов испытаний после интеграции на соответствие требованиям - Валидация системы программного обеспечения для принятия заказчиком (потребителем) - Регрессионная стратегия повторного тестирования изменений в интеграции программного обеспечения - Документирование результатов приемки квалификации	- Проведение испытаний на повышение надежности и ускоренных испытаний по мере необходимости - Мониторинг тенденций и улучшения надежности - Выполнение квалификационных испытаний - Анализ результатов испытаний для приемки - Инициация приемки потребителем (заказчиком) - Валидация системы программного обеспечения на соответствие требованиям заказчика, включая демонстрацию показателей надежности работы и функций безопасности, защищенности и целостности, где это применимо - Подготовка документа о статусе версии программного обеспечения

В.9 Эксплуатация и техническое обслуживание

Требования системы к программному обеспечению	Соответствующие действия по обеспечению надежности
- Процедуры и условия эксплуатации - Стратегия поддержки сопровождения - Логистическая поддержка - Сбор данных эксплуатации - Обучение пользователей - Программа гарантии на программное обеспечение для поддержания надежности системы в эксплуатации	- Мониторинг тенденций работы в эксплуатации - Обновление записей о работе и сопровождении в эксплуатации - Проведение опросов об удовлетворенности потребителей - Анализ данных эксплуатации для определения области повышения безотказности - Установление профиля эксплуатации - Ведение базы данных о результатах испытаний и эксплуатации системы и ее прототипов - Сбор соответствующих показателей надежности для прогнозирования безотказности - Внедрение лучших практик гарантии на программное обеспечение

В.10 Обновление/улучшение программного обеспечения

Требования системы к программному обеспечению	Соответствующие действия по обеспечению надежности
- Обновление программного обеспечения - Безупречное выполнение стратегии обслуживания - Введение новой услуги и оценка ее влияния - Влияние улучшения на работу программного обеспечения	- Отслеживание обновления программного обеспечения - Безупречное выполнение обслуживания - Выполнение изменений проекта и контроль конфигурации - Оценка влияния введения новых услуг - Управление выпуском новых версий программного обеспечения

B.11 Вывод из эксплуатации

Требования системы к программному обеспечению	Соответствующие действия по обеспечению надежности
- Прекращение выполнения конкретной услуги - Консультирование пользователей о прекращении выполнения старых услуг и замене их новыми.	- Информирование пользователей о прекращении использования программного обеспечения и выполнения услуг поддержки - Консультирование клиентов о любых необходимых действиях по обеспечению надежности

Приложение С
(справочное)

Процесс комплексной модели зрелости

Комплексная модель зрелости (CMMI) — это процесс улучшения модели зрелости для обеспечения надежности программных продуктов и услуг. Процесс состоит из лучших практик, которые касаются деятелей разработки и технического обслуживания, охватывающих весь жизненный цикл продукта от концепции до поставки и технического обслуживания. CMMI для разработки модулей [9] содержит методы, охватывающие управление проектами, управление процессами, разработку системы, разработку аппаратных средств, разработку программных средств и другие вспомогательные процессы, используемые при разработке и обслуживании. Процесс CMMI коррелирует с выполнением требований к системе программного обеспечения и соответствующими действиями по обеспечению надежности, как показано в приложении В. CMMI используют для бенчмаркинга и оценки, а также для руководства усилиями организации по постоянному улучшению. Процесс CMMI относят к одному из уровней.

- Уровням возможностей, которые относятся к непрерывному представлению, их применяют к достижению организацией улучшения процессов в отдельных областях процессов. Эти уровни являются для руководства процессом средствами постепенного улучшения, соответствующими данной области процесса. Существует шесть уровней возможностей, пронумерованных от 0 до 5.
- Уровням зрелости, которые относятся к поэтапному представлению, их применяют к достижениям улучшения процессов организации в нескольких областях процессов. Эти уровни используют для прогнозирования общих результатов следующего проекта. Существуют пять уровней зрелости, пронумерованных от 1 до 5.

В таблице С.1 шесть уровней возможностей приведены в соответствие с пятью уровнями зрелости для сравнения.

Таблица С.1 — Сравнение уровней возможностей и зрелости

Уровень	Уровни возможностей	Уровни зрелости
0	Неполный процесс — это процесс, который либо не выполняют, либо выполняют частично. Одна или несколько конкретных целей процесса не достигнуты. Для этого уровня не существует общих целей, поскольку нет оснований формализовать частично выполняемый процесс	Нет данных
1	Выполняемый процесс — это процесс, который удовлетворяет установленным целям к процессу. Он поддерживает и делает возможным работу, необходимую для создания рабочих продуктов. Хотя уровень возможностей 1 приводит к важным улучшениям, эти улучшения могут быть потеряны со временем, если они не установлены. Установление помогает обеспечить поддержку улучшений	На уровне зрелости 1 процессы обычно носят случайный и хаотичный характер. Организация обычно не обеспечивает стабильные условия для поддержки процессов. Успех в этих организациях зависит от компетентности специалистов организации, а не от использования установленных процессов. Организации, имеющие уровень зрелости 1, часто производят работоспособные продукты и услуги, однако такие организации часто превышают запланированный бюджет и не выполняют графики выполнения работ. Существует тенденция к принятию чрезмерных обязательств, отказу от процессов в случае кризиса и неспособности воспроизведения успешных проектов

Продолжение таблицы С.1

Уровень	Уровни возможностей	Уровни зрелости
2	Управляемый процесс — это выполняемый процесс, который имеет базовую инфраструктуру для поддержки данного процесса. Этот процесс планируют и выполняют в соответствии с политикой организации; нанимают квалифицированных специалистов, обладающих достаточными ресурсами для получения управляемых результатов; вовлекают соответствующие заинтересованные стороны; проводят мониторинг, управляют, анализируют и оценивают на предмет соответствия описанию процесса. Организация процесса в соответствии с уровнем возможностей 2 помогает гарантировать сохранение существующих практик в кризисный период	На уровне зрелости 2 при выполнении проектов организации обеспечивают планирование и выполнение процессов в соответствии с политикой; в проектах работают квалифицированные специалисты, обладающие достаточными ресурсами для получения управляемых результатов; в работу вовлекают соответствующие заинтересованные стороны; проводят мониторинг, управляют, анализируют и оценивают на предмет соответствия описанию процесса. Результаты уровня зрелости организации 2 обеспечивают сохранение существующей практики в кризисные периоды; выполнение проектов и управление ими в соответствии с их документированными планами; состояние разрабатываемых продуктов и предоставления услуг демонстрируют руководству в определенные моменты на основных этапах и по завершении основных задач. Соответствующие заинтересованные стороны устанавливают между собой обязательства и пересматривают их по мере необходимости. Разрабатываемые продукты должным образом управляются Разрабатываемые продукты и услуги удовлетворяют установленным описаниям процессов, стандартам и процедурам.
3	Определенный процесс — это управляемый процесс, который адаптирован в соответствии с набором стандартных процессов организации, руководящими принципами организации, процесс вносит разрабатываемые показатели, меры и другую информацию по совершенствованию процессов в активы процесса организации. Соответствующие стандарты, описания процессов и процедуры согласованы и адаптированы к конкретному проекту или к подразделению организации. Определенный процесс четко устанавливает цель, входные данные, входные критерии, виды деятельности, обязанности, параметры, этапы проверки, результаты и выходные критерии. Процессами управляют проактивно, путем понимания взаимосвязей действий процесса и показателей процесса, разрабатываемых им продуктов и услуг	На уровне зрелости 3 процессы организации хорошо характеризованы и понятны, а также описаны в стандартах, процедурах и методах. Эти стандартные процессы используют для установления их последовательности при выполнении в организации. Проекты устанавливают свои определенные процессы путем адаптации набора стандартных процессов организации в соответствии с руководящими принципами адаптации. Результаты организаций, имеющих уровни зрелости 3, демонстрируют постоянство в работе

Окончание таблицы С.1

Уровень	Уровни возможностей	Уровни зрелости
4	Количественно управляемый процесс — это процесс, которым управляют с помощью статистических и других количественных методов. Устанавливают количественные цели качества и работы процесса, которые используют в качестве критериев управления процессом. Качество и работа процесса понимаются в статистических терминах и управляются на протяжении всего жизненного цикла процесса с помощью статистических методов	На уровне зрелости организации 4 для проектов устанавливают количественные цели качества и эффективности процессов и используют их в качестве критериев управления процессами. Количественные цели основаны на потребностях заказчика (потребителя), конечных пользователей, организации и исполнителей процессов. Качество и работу процессов понимают в виде статистических показателей, которыми управляют на протяжении всего жизненного цикла процессов. Для отобранных подпроцессов собирают и статистически анализируют подробные показатели работы процесса. Показатели качества и работы процессов включены в хранилище показателей организации для поддержки принятия решений на основе фактов. Выявляют особые причины изменения процесса и, при необходимости, корректируют источники особых причин для предотвращения появления их в будущем. Результаты деятельности организаций 4-го уровня зрелости демонстрируют адекватное управление работой процессов с использованием статистических и других количественных методов для обеспечения количественной предсказуемости результатов работы
5	Оптимизирующий процесс — это количественно управляемый процесс, который совершенствуется на основе понимания общих причин изменчивости, присущей процессу. Основное внимание в процессе оптимизации уделяют постоянному улучшению работы процесса за счет постоянных инновационных улучшений	На 5-м уровне зрелости организация постоянно совершенствует свои процессы, основываясь на количественном понимании общих причин изменчивости, присущей процессам; фокусируется на постоянном улучшении работы процессов путем постепенного и инновационного совершенствования процессов и технологий. Устанавливают количественные цели улучшения процессов. Их постоянно пересматривают с учетом меняющихся бизнес-целей и используют в качестве критериев управления улучшением процессов. Результаты внедренных улучшений процессов измеряют и оценивают с учетом количественных целей улучшения процесса. Как определенные процессы, так и набор стандартных процессов организации являются целями измеримых мероприятий по улучшению. Результаты организаций 5-го уровня зрелости демонстрируют постоянное улучшение процессов для достижения установленных количественных целей улучшения процессов

Приложение D
(справочное)

Классификация признаков дефектов программного обеспечения

D.1 Общие положения

Ортогональная классификация дефектов (ODC) — это метод, используемый для сбора и группировки информации о неисправностях программного обеспечения в терминах свойств дефектов программного обеспечения. Процесс ODC обеспечивает возможность выявления отличительных признаков дефектов и определения работоспособности процесса разработки программного обеспечения. Классификация основана на том, что известно о дефекте. Если неисправность или дефект обнаружены, то обычно известен и способ, которым дефект был обнаружен, и его влияние на пользователя. Таким образом, могут быть классифицированы особенности ODC, такие как действие, триггер и воздействие. Точно так же, если неисправность диагностирована и устранена, детали восстановления известны. Могут быть классифицированы особенности ODC, такие как цель, тип, квалификатор, источник и возраст дефекта. Дополнительные особенности, не относящиеся к ODC, такие как плановый этап проекта (на котором был найден дефект), значимость и компонент, которые фиксируют в любой системе отслеживания неисправностей или дефектов, могут быть использованы вместе с анализом на основе ODC. ODC не навязывает конкретную структуру. Ниже сделано обобщение классификации особенностей дефектов программного обеспечения на открытие, закрытие и действия триггера.

D.2 Открытие

Открытие предназначено для классификации особенностей при обнаружении неисправности.

Когда неисправность обнаружена и открыта для диагностики в процессе разработки программного обеспечения, может быть оценена информация о выявлении неисправности, ее вероятном воздействии и степени тяжести. Типичные сведения о неисправностях, собираемые при обнаружении неисправности, приведены в таблице D.1.

Таблица D.1 — Классификация особенностей дефектов программного обеспечения при обнаружении неисправности

Действия[a] по устранению неисправностей (при обнаружении)	Триггер[b] (обнаружения)	Воздействие[c] (влияние и значимость)
- Анализ проекта - Проверка кода - Тестирование модулей - Тестирование функций - Тестирование системы - Приемочные испытания - Квалификационные испытания - Испытания на повышение надежности	- Соответствие проекту - Совместимость - Параллельность - Охват - Последовательность - Взаимодействие - Конфигурация	- Возможность установки - Удобство обслуживания - Целостность/безопасность/защищенность - Безотказность - Обслуживание - Доступность - Удобство использования
Примечание 1 — Неисправности программного обеспечения часто трудно воспроизвести. Важно зафиксировать обстоятельства и условия, приводящие к неисправности. Примечание 2 — Требуется прослеживаемость требований; либо прослеживаемость требований к программному обеспечению, либо прослеживаемость конкретного тестового примера.		
[a] Действия: фактические действия, выполняемые в момент обнаружения неисправности. [b] Триггер: среда или условие, которые должны существовать для появления неисправности. [c] Воздействие: для неисправностей разработки воздействие оценивают как возможное влияние и значимость для пользователя; для неисправностей, зарегистрированных в эксплуатации, воздействие — это последствие и значимость отказа для пользователя.		

D.3 Закрытие

Закрытие предназначено для классификации особенностей устранения неисправности.

Когда неисправность устранена, то известны точный характер неисправности и объем необходимых исправлений. Типичные сведения о неисправностях, собираемые при устранении неисправности, приведены в таблице D.2.

Таблица D.2 — Классификация сведений о дефектах программного обеспечения при устранении неисправности

Цель[a]	Тип[b]	Квалификатор[c]	Возраст[d]	Источник[e]
Проект/код	Инициация Проверка Функция Синхронизация Интерфейс	Отсутствие Некорректность Несовместимость	Основной Новый Улучшенный	Разработано внутри компании Разработано при помощи аутсорсинга Повторно использован из библиотеки Портирован

[a] Цель: верхнеуровневая идентификация объекта исправления.

[b] Тип: особенности фактической коррекции, которая была сделана.

[c] Квалификатор: (применяется к типу дефекта): описывает элемент либо отсутствия, либо некорректности, либо ошибки, либо несовместимого выполнения.

[d] Возраст: идентифицирует историю цели (проект/код), в которой обнаружен дефект.

[e] Источник: определяет происхождение цели (проект/код), которая имела дефект.

D.4 Составление карты триггеров и действий

Составление карты действий и триггеров группирует применимые триггеры, относящиеся к деятельности, связанной с анализом контроля и тестированием проекта программного обеспечения. В таблицах D.3, D.4, D.5 и D.6 приведено несколько общих примеров составления карт действий и триггеров.

Таблица D.3 — Составление карты действий и триггеров для анализа проекта и контроля кодов

Действия	Триггеры
Анализ проекта/ контроля/кодирования Пересмотр проекта или сравнение проектной документации с известными требованиями	Соответствие проекта Проверяющий инспектор обнаруживает неисправность, сравнивая проверяемый элемент проекта или сегмент кода с его спецификацией, установленной на предыдущем этапе. Проверка может охватывать проектную документацию, кодекс, практику разработки и стандарты, а также гарантировать, что требования проекта не будут пропущены или не будут неопределенными. Логика/поток Инспектор использует знание основных методов программирования и стандартов для изучения логики потока или данных, чтобы гарантировать их правильность и полноту. Обратная совместимость Инспектор использует обширный опыт работы с продуктом/компонентом для выявления несовместимости функции, описанной в проектной документации или кодах, и более ранними версиями того же продукта или компонента. С точки зрения эксплуатации приложение пользователя, которое успешно работало в предыдущем выпуске, отказывает в текущем выпуске Горизонтальная совместимость Инспектор, обладающий обширным опытом, обнаруживает несовместимость функции, описанной в проектной документации или кодах, и другими системами, продуктами, услугами, компонентами или модулями, с которыми функция должна взаимодействовать. Параллельность Инспектор рассматривает организацию серийного производства, необходимую для управления общим ресурсом при обнаружении неисправности. Это включает в себя организацию серийного выпуска большого количества функций, потоков, процессов или контекстов ядра, а также наложение и снятие блокировок. Внутренний документ Наличие неверной информации, несоответствий или неполноты во внутренней документации. Прологи, комментарии к коду и планы тестирования представляют собой некоторые примеры документации, которая подпадает под эту категорию

Окончание таблицы D.3

Действия	Триггеры
	Языковая зависимость Разработчик обнаруживает дефект при проверке специфических для языка деталей выполнения компонента или функции. Стандарты языка, проблемы компиляции и эффективность конкретных языков являются примерами проблемных областей Побочные эффекты Инспектор использует обширный опыт и знание продукта, чтобы предвидеть поведение какой-либо системы, продукта, функции или компонента, которое может возникнуть в результате рассматриваемого проекта или кода. Побочные эффекты характеризуют результат общего использования или конфигураций, в том числе за пределами области действия компонента или функции, с которыми связан рассматриваемый проект или код Редкая ситуация Инспектор использует обширный опыт и знание продукта, чтобы предвидеть некоторое поведение системы, которое не учитывается или не рассматривается в исследуемом документированном проекте или коде и обычно связано с необычными конфигурациями или использованием. Отсутствующее или неполное восстановление неисправностей, как правило, не классифицируют как триггер редкой ситуации, но, скорее всего, относят к соответствию конструкции, если неисправность обнаружена во время пересмотра/проверки

Таблица D.4 — Карты действий и триггеров для тестирования модулей

Действия	Триггеры
Тестирование модулей Тестирование белого ящика на основе детального знания внутренних компонентов кода	Простой путь Тестовый пример, мотивированный знанием конкретных ветвей кода, а не внешним знанием функциональности. Этот триггер обычно не выбирают для записей о дефектах эксплуатации, кроме случаев, когда потребитель очень хорошо осведомлен о внутреннем коде и проекте и специально вызывает определенный путь (как это иногда бывает, когда потребитель является деловым партнером или поставщиком программного обеспечения). Сложный путь При тестировании белого/серого ящика в тестовом примере обнаружен дефект, связанный с выполнением некоторых надуманных комбинаций путей кода. Тестировщик пытается вызвать выполнение нескольких ветвей кода в нескольких различных условиях. Этот триггер выбирают для зарегистрированных дефектов в эксплуатации только в тех же обстоятельствах, что и описанные в разделе «Простой путь»

Таблица D.5 — Карты действий и триггеров для тестирования функций

Действия	Триггеры
Тестирование функций Выполнение тестирования черного ящика на основе внешних спецификаций функций	Охват Во время тестирования черного ящика тестовый пример, который выявил дефект, был попыткой прямого выполнения кода для единственной функции, без использования параметров или с использованием единственного набора параметров. Изменение Во время тестирования черного ящика тестовый пример, обнаруживший дефект, был попыткой выполнения кода для единственной функции, но с использованием различных входных данных и параметров. Они могут включать недопустимые параметры, экстремальные значения, граничные условия и комбинации параметров. Последовательность действий Во время тестирования черного ящика тестовый пример, обнаруживший дефект, выполнял несколько функций в очень определенной последовательности. Этот триггер выбирают только тогда, когда каждая функция успешно выполняется при независимом запуске, но отказывает в этой конкретной последовательности. Также может быть успешно выполнена другая последовательность.

Окончание таблицы D.5

Действия	Триггеры
	Взаимодействие Во время тестирования черного ящика тестовый пример, обнаруживший дефект, инициировал взаимодействие двух или более тел кода. Этот триггер выбирают только тогда, когда каждая функция успешно выполняется при независимом запуске, но отказывает в этой конкретной комбинации. Взаимодействие включает в себя нечто большее, чем просто набор последовательностей выполнений

Таблица D.6 — Карты действий и триггеров для тестирования системы

Действия	Триггеры
Тестирование системы Тестирование или запуск полной версии системы в реальных условиях, требующих всех ресурсов	Рабочая нагрузка/повышенная нагрузка Система работает на некотором пределе ресурсов, верхнем или нижнем. Эти ограничения ресурсов могут быть созданы с помощью различных механизмов, включая выполнение малых или больших нагрузок, нескольких продуктов одновременно, работу системы в течение длительного периода времени. Восстановление/исключение Систему тестируют с целью активации обработчика исключений или какого-либо типа кода восстановления. Дефект не проявился бы, если бы не произошел какой-то более ранний вызов обработки исключений или восстановления. С точки зрения обнаружения дефектов в условиях эксплуатации этот триггер мог быть выбран, если бы был обнаружен дефект в способности системы или продукта быть восстановленным после отказа. Запуск/перезапуск Система или подсистема инициализировалась или перезапускалась после некоторого более раннего завершения работы или отказа системы в целом или подсистемы. Конфигурация аппаратных средств Систему тестируют для обеспечения корректного выполнения функций в определенных конфигурациях аппаратного обеспечения. Конфигурация программного обеспечения Систему тестируют для обеспечения корректного выполнения функций в определенных конфигурациях программного обеспечения. Блокированный тест (нормальный режим) Продукт работает хорошо в пределах ресурсов, а дефект обнаруживают при попытке выполнить сценарий тестирования системы. Этот триггер используют, когда сценарии не могут быть запущены из-за базовых проблем, которые препятствуют их выполнению. Этот триггер не должен быть использован для дефектов, обнаруженных заказчиком (потребителем)

Приложение Е
(справочное)

Примеры данных программного обеспечения, полученных в результате сбора данных

Е.1 Данные о неисправностях

Данные	Применение
Данные записей и отчетов о проблемах: - дата и время обнаружения неисправности; - описание обнаруженной неисправности; - неисправность обнаружена в программной области; - специалист, обнаруживший неисправность; - симптомы и состояния неисправности; - значимость и приоритет	Данные, собранные по проектам программного обеспечения, должны быть использованы в отчете о проблеме после возникновения и идентификации неисправности
Данные о корректирующих действиях: - дата устранения неисправности; - специалист, исправивший ошибку; - выполнение действий технического обслуживания; - описание модификации; - идентификация модифицированных модулей; - информация о контроле версий; - время, необходимое для устранения неисправности; - дата верификации устранения неисправности; - специалист, верифицировавший устранение неисправности	Собранные данные о корректирующих действиях, проверенные как исправленные неисправности, должны использоваться для составления отчетов об устранении проблем
Накопленные данные об обнаруженных неисправностях	Накопленные данные об обнаруженных неисправностях следует использовать для определения интенсивности возникновения неисправностей и тенденции изменения вероятности безотказной работы в течение определенного периода времени.
Накопленные данные об устраненных неисправностях	Накопленные данные об устраненных неисправностях следует использовать для определения известных неисправностей, требующих корректирующих действий, и отслеживания эффективности действий технического обслуживания
Интенсивность обнаружения неисправностей	Интенсивность обнаружения неисправностей используют для выявления тенденций и облегчения планирования стратегии технического обслуживания и управления ресурсами
Интенсивность устранения неисправностей	Интенсивность устранения неисправностей используют для выявления тенденций и облегчения планирования стратегии технического обслуживания и управления ресурсами. Установка приоритета для действий технического обслуживания основана на значимости проблем, вызванных неисправностью
Неисправности по расположению	Отслеживание неисправностей в соответствии с программными функциями для идентификации конкретных областей кода, более подверженных ошибкам
Критичность неисправностей	Классификация степени воздействия неисправностей для установления приоритета действий технического обслуживания

Окончание таблицы Е.1

Данные	Применение
Количество и процент значимых неисправностей	Показания к планированию стратегии технического обслуживания.
Структурная сложность в соответствии с расположением	Используется с другими показателями для определения влияния генерируемых неисправностей, связанных со сложностью структуры программного обеспечения и расположением
Функциональная сложность в соответствии с расположением	Используется с другими показателями для определения влияния генерируемых неисправностей, связанных со сложностью функций программного обеспечения и с расположением (в коде)

Е.2 Данные о продукте

Данные	Применение
Количество и процент модулей, выполняющих более одной функции	Индексация совместимости проекта программного обеспечения в целом по функциональной сложности. Высокая сложность модуля ведет к низкой совместимости, что требует повторного проектирования
Количество и процент модулей, имеющих высокую структурную сложность	Указание на необходимость перепроектирования программного обеспечения в целом для снижения сложности
Количество и процент модулей, имеющих ровно один вход и один выход	Указание совместимости проекта, которое должно быть использовано в качестве основы при структурированном проектировании
Количество и процент модулей, документированных в соответствии со стандартами	Указание полноты кода, которое следует использовать для определения того, что код содержит все требования и полностью им удовлетворяет
Количество и процент неисправностей, обнаруженных в повторно используемом коде	Указывает на ненадежность повторно используемого кода

Е.3 Данные о процессе

Данные	Применение
Неисправности, внесенные на стадиях жизненного цикла	Указание того, когда и на каком этапе были внесены неисправности и приняты соответствующие меры
Неисправности, обнаруженные на стадиях жизненного цикла	Указание того, когда и на какой стадии были обнаружены неисправности, а также обоснование задержки корректирующих действий по устранению неисправности
Общее время, затраченное на анализ	Указание времени, затраченного на анализ, для выявления проблемы и определения корректирующих действий, а также соответствующих необходимых ресурсов
Общее время, затраченное на проектирование	Указание времени, затраченного на проектирование программного обеспечения, и соответствующих необходимых ресурсов
Общее время, затраченное на кодирование	Указание времени, затраченного на кодирование и программирование, и соответствующих необходимых ресурсов
Общее время, затраченное на тестирование модулей	Указание времени, затраченного на тестирование модулей, и соответствующих необходимых ресурсов
Общее время, затраченное на тестирование системы	Указание времени, затраченного на тестирование системы, и соответствующих необходимых ресурсов

Окончание таблицы Е.3

Данные	Применение
Общее время технического обслуживания	Указание времени, затраченного на техническое обслуживание, и соответствующих необходимых ресурсов
Среднее время администрирования технического обслуживания	Указание времени, затраченного на администрирование технического обслуживания, и соответствующих необходимых ресурсов. Административные обязанности по техническому обслуживанию включают в себя действия до и после устранения неисправности, такие как время, затраченное на назначение обслуживающего персонала, выпуск новой (исправленной) версии
Среднее время корректирующих действий	Указание времени, затраченного на корректирующие действия, и соответствующие необходимые ресурсы. Отражает экономическую эффективность деятельности по техническому обслуживанию
Причина корректирующих действий	Используют для определения источника неисправностей Типичные причины включают в себя: - предыдущие действия технического обслуживания; - новые требования; - изменение требований; - неверно истолкованное требование; - отсутствие требования; - неоднозначное требование; - изменение программной среды; - изменение аппаратной среды; - ошибка в коде или логическая ошибка; - ошибка при работе
Стоимость корректирующих действий	Указание общей стоимости корректирующих действий, включая обнаружение неисправностей, решение проблем и администрирование для эффективного технического обслуживания
Процент тестированных и верифицированных функций	Индикация охвата тестированием, эффективности и полноты тестирования
Процент тестированных и верифицированных независимых путей	Указание охвата тестированием структурных испытаний и полноты их проведения
Процент тестированных и верифицированных исходных строк кода	Указание охвата тестированием программного кода и его полноты
Исторические данные	Предоставление предыдущих данных по проблемным областям, связанным с проектированием, процессами и продуктами

Приложение F
(справочное)

Пример функций надежности комбинированного аппаратного и программного обеспечения

Система управления мониторингом представляет собой комбинированную программно-аппаратную систему, состоящую из электронных операций и прикладных функций, представленных структурой системы. Ниже приведены основные описания функций аппаратного и программного обеспечения системы:

- аппаратная функция срабатывания для работы управляющего механизма мониторинга;
- компоненты электронной схемы срабатывания;
- аппаратная функция распознавания активации запуска или остановки;
- компоненты электронной схемы распознавания;
- функции применения программного обеспечения для управления быстрым запуском и отключением аппаратной функции распознавания, включая:
  - программный модуль управления инициированием активации,
  - программный модуль управления отключением и деактивацией,
  - программный модуль для мониторинга скорости активации и деактивации.

Эти функции представлены схемой, показанной на рисунке F.1.

Рисунок F.1 — Структурная схема системы управления мониторингом

Аппаратную функцию срабатывания можно рассматривать как аппаратную подсистему с полностью электронной схемой срабатывания, состоящую из электронных компонентов.

Аппаратную функцию распознавания можно рассматривать как аппаратную подсистему с полностью электронной схемой распознавания, состоящей из электронных компонентов.

Функции применения программного обеспечения состоят из трех отдельных программных модулей, предназначенных для совместной работы с аппаратной подсистемой распознавания. Каждый программный модуль предназначен для выполнения только одной функции согласно назначению. Для того чтобы подсистема аппаратного распознавания работала, необходимо, чтобы все программные модули были включены в подсистему применения программного обеспечения и работали как объединенная аппаратно-программная подсистема.

Для анализа безотказности при определении интенсивности отказов систем и подсистем следует учитывать следующее. В основе анализа лежит предположение о постоянной интенсивности отказов электронных компонентов, которые охватывают все аппаратные функции.

- Интенсивность отказов аппаратной функции срабатывания может быть определена с помощью суммы интенсивностей отказов отдельных компонентов схемы срабатывания при отсутствии в конструкции резервирования и непрерывной работе в течение полного рабочего времени (λ_1).

- Интенсивность отказов аппаратной функции распознавания может быть определена с помощью суммы интенсивностей отказов отдельных компонентов схемы распознавания, при отсутствии в конструкции резервирования и непрерывной работе полный рабочий день (λ_2).

- Вероятность безотказной работы функций применения программного обеспечения определяет общую интеграцию системы из трех программных модулей с их зависимостью, связанной с профилем эксплуатации системы управления. Профиль эксплуатации не требует полного календарного времени работы этих программных модулей, а только их частичное применение по запросу. Время выполнения, соответствующее каждому программному модулю, должно быть включено в расчет плотности распределения неисправностей, относящейся к каждому соответствующему программному модулю. Затем плотность распределения отказов преобразуют в календарное время для определения эффективной интенсивности отказов. Плотность отказов используют для восстанавливаемых объектов, а интенсивность отказов — для невосстанавливаемых объектов. Поскольку программное обеспечение

не подлежит ремонту, здесь использована интенсивность отказов. В этом отношении функции применения программного обеспечения рассматривают как один элемент конфигурации программного обеспечения с интенсивностью отказов (λ_3).

- Вероятность безотказной работы системы и управления мониторингом с точки зрения интенсивности отказов соответствует сумме $\lambda_1 + \lambda_2 + \lambda_3 = \lambda$.

Приложение G
(справочное)

Краткое описание модели безотказности программного обеспечения

Существует большое количество моделей безотказности программного обеспечения, существующих сегодня и используемых в отраслевой практике. Большая часть моделей разработана для конкретного применения. В основном они компьютеризированы и автоматизированы для облегчения обработки, анализа и оценки данных. Не существует единой модели, которая подходила бы для всех вариантов применения. Пользователь сам выбирает и применяет соответствующие модели безотказности программного обеспечения для удовлетворения своих потребностей проектирования. Ниже представлен перечень параметров, используемых в большей части моделей безотказности программного обеспечения.

a) Общее количество неисправностей, присущих программному обеспечению. Предполагается, что это количество является постоянным и конечным.

b) Общее количество скрытых неисправностей (ошибок) в программном обеспечении. Предполагается, что это количество является переменным из-за возможности введения в код новых неисправностей с течением времени.

c) Общее количество неисправностей, устраненных в определенный момент времени или по истечении некоторого времени использования или тестирования.

d) Общее количество неисправностей, обнаруженных в определенный момент времени или по истечении некоторого времени использования или тестирования.

e) Количество периодов или интервалов тестирования. Это количество периодов времени между действиями по устранению неисправностей. Некоторые модели предполагают, что неисправности устраняют сразу же после их обнаружения.

f) Общее количество неисправностей, устраненных за определенный период тестирования (испытаний).

g) Изменение интенсивности отказов.

h) Время тестирования или использования, накопленное до настоящего времени, или текущее количество обнаруженных неисправностей.

i) Накопленное время работы.

j) Начальная интенсивность отказов.

k) Настоящая (текущая) интенсивность отказов.

l) Повышение интенсивности.

m) Оценка количества строк исполняемого кода во время тестирования или использования.

n) Общее количество выполненных тестовых примеров.

o) Общее количество успешно выполненных тестовых примеров.

Приложение Н
(справочное)

Выбор и применение моделей безотказности программного обеспечения

В настоящее время существует много моделей и показателей для оценки вероятности безотказной работы и оценки характеристик программного обеспечения. Все модели разработаны для подгонки кривых модели с использованием входных данных. Достоверность и точность применения модели и полученного результата зависят от предположений, сделанных при построении модели, и релевантности входных данных модели. Наибольшее количество моделей разработано для удовлетворения конкретных потребностей в течение жизненного цикла программного обеспечения. Примеры включают модель прогнозирования в процессе проектирования программного обеспечения и модель оценки дополнительного времени тестирования, необходимого перед выпуском программного обеспечения. Некоторые модели разработаны для прогнозирования надежности программного обеспечения еще до написания кода. Ввод данных для прогнозирования надежности в таком случае часто основывается на данных работы и применения аналогичной программной системы. Другие модели используют для оценки тенденций повышения интеграции системы программного обеспечения, на основе промежуточных тестовых входных данных. Не существует единой модели, способной охватить весь жизненный цикл программного обеспечения. На практике для определения вероятности безотказной работы программного обеспечения часто используют несколько моделей. Для выбора модели часто используют статистические методы, такие как критерии согласия, чтобы проверить, насколько хорошо модель соответствует набору наблюдений. Большая часть моделей для оценки вероятности безотказной работы программного обеспечения автоматизированы из-за их итерационных вычислений. Интерпретация результатов моделирования безотказности требует практического опыта и знаний в области надежности.

В таблице Н.1 представлены некоторые примеры моделей безотказности программного обеспечения, используемых в отраслевой практике. В цели настоящего стандарта не входит детальная формулировка модели и ее применения. Ссылки на модели безотказности программного обеспечения и их конкретные применения хорошо описаны в литературе [14, 37].

Таблица Н.1 — Примеры моделей безотказности программного обеспечения

	Наименование модели	Предположения	Требования к данным	Ограничения применения
1	Musa-basic	- Конечное количество присущих неисправностей (скрытых неисправностей) - Постоянная интенсивность неисправностей во времени - Экспоненциальное распределение	- Количество обнаруженных неисправностей в определенный момент времени - Оценка начальной интенсивности отказов - Текущая интенсивность отказов системы программного обеспечения	- Программное обеспечение находится в эксплуатации - Использование после интеграции системы - Предполагается, что корректировка не вводит новые неисправности - Предполагается, что количество остаточных неисправностей линейно уменьшается с течением времени
2	Musa-Okumoto	- Бесконечное количество присущих неисправностей (скрытых неисправностей) - Изменение интенсивности ошибок во времени - Логарифмическое распределение	- Количество обнаруженных неисправностей в определенный момент времени - Оценка начальной интенсивности отказов - Относительное изменение интенсивности отказов во времени - Текущая интенсивность отказов системы программного обеспечения	- Программное обеспечение находится в эксплуатации - Использование модулей в тестовых системах - Предположение, что корректировка не вводит новые неисправности - Предположение, что количество остаточных неисправностей со временем экспоненциально уменьшается

Продолжение таблицы Н.1

	Наименование модели	Предположения	Требования к данным	Ограничения применения
3	Jelinski-Moranda	- Конечное и постоянное количество присущих неисправностей (скрытых неисправностей) - Постоянная интенсивность ошибок во времени - Неисправности устраняют сразу после обнаружения - Биномиальное экспоненциальное распределение	- Количество устраненных неисправностей в определенный момент времени - Оценка начальной интенсивности отказов - Текущая интенсивность отказов системы программного обеспечения	- Программное обеспечение находится в эксплуатации - Использование после интеграции системы - Предположение, что корректировка не вводит новые неисправности - Предположение, что количество остаточных неисправностей линейно уменьшается с течением времени
4	Littlewood-Verrall	Неопределенность в процессе коррекции	- Оценка количества отказов - Оценка скорости повышения надежности - Время между обнаруженными отказами или время возникновения отказа	Программное обеспечение находится в эксплуатации
5	Schneidewind	Корректировка не вводит новые неисправности	- Оценка интенсивности отказов в начале первого интервала - Оценка константы пропорциональности изменения интенсивности отказов во времени - Неисправности, обнаруженные в равные промежутки времени	- Программное обеспечение находится в эксплуатации - Скорость обнаружения неисправностей уменьшается экспоненциально с течением времени
6	Geometric	Количество присущих неисправностей должно быть бесконечным	- Уменьшение в соответствии с функцией геометрической прогрессии по мере обнаружения отказов - Время между возникновением отказов или время возникновения отказа	- Программное обеспечение находится в эксплуатации - Неисправности независимы и неравны по вероятности возникновения и степени тяжести
7	Brooks-Motley	Интенсивность обнаружения неисправностей во времени	- Трудозатраты выполнения каждого теста - Вероятность обнаружения неисправности в *i*-м тесте - Вероятность устранения неисправностей без введения новых - Количество неисправностей, оставшихся к началу *i*-го теста - Общее количество неисправностей, обнаруженных в каждом тесте	- Программное обеспечение в процессе разработки - Некоторые программные модули отличаются по тестовым усилиям

Окончание таблицы Н.1

	Наименование модели	Предположения	Требования к данным	Ограничения применения
8	Bayesian	Программное обеспечение относительно свободно от неисправностей	- Продолжительность времени тестирования для каждого интервала - Количество отказов, обнаруженных в каждом интервале	- Программное обеспечение находится в эксплуатации - Программное обеспечение корректируется в конце интервала тестирования
9	Keene	Корреляция количества скрытых неисправностей в поставляемых блоках с возможностями процесса разработки и объемом программного обеспечения (KSLOCs)	Уровень зрелости CMM для оценки возможностей процесса, оцененный KSLOC поставляемый код, оцененное количество месяцев до достижения зрелости после выпуска, скрытые неисправности, процент неисправностей 1-й и 2-й степени значимости, время восстановления, часы использования кода в неделю и процент активации неисправностей	- Процент активации неисправностей является оценочным параметром, представляющим средний процент мест пользователей системы, у которых, вероятно, возникнет конкретная неисправность - Уровень возможностей CMM должен быть оценен для организации разработки, а также для организации технического обслуживания

Для облегчения выбора модели следует использовать следующие критерии:
- профили отказов;
- зрелость программного продукта;
- параметры разработки программного обеспечения;
- характеристики тестирования программного обеспечения;
- существующие показатели и данные.

Для выполнения модели рекомендуется использовать автоматизированные вычислительные средства. Существуют коммерчески доступные средства, которые охватывают некоторые или все модели безотказности программного обеспечения, указанные в таблице Н.1. Основным преимуществом использования автоматизированных вычислительных средств является экономия времени и затрат на реализацию модели. Выбрав подходящий метод, можно сравнить результаты набора данных, выполненных на нескольких моделях, чтобы определить наиболее подходящий.

При выборе метода организации следует учитывать следующие критерии:
- наличие средства, совместимого с компьютерными системами организации;
- стоимость установки и обслуживания программы;
- количество исследований, которые, вероятно, будут проведены для применения средства;
- типы систем программного обеспечения, подлежащих изучению;
- качество документации о применении средства;
- простота изучения средства;
- гибкость и мощь средства;
- техническая поддержка средства.

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 60050-191:1990*	—	Международный стандарт включает текст на русском языке
IEC 60300-3-15:2009	MOD	ГОСТ Р 27.015—2019 «Надежность в технике. Управление надежностью. Руководство по проектированию надежности систем»

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

- MOD — модифицированный стандарт.

* IEC 60050-191:1990 заменен на IEC 60050-192:2015, которому соответствуют национальные стандарты ГОСТ Р 27.101—2021 (NEQ) и ГОСТ Р 27.102—2021 (NEQ).

Библиография

[1] IEC 62508, Guidance on human aspects of dependability

[2] ISO/IEC 12207, Systems and software engineering — Software life cycle processes

[3] IEC 60300-1, Dependability management — Part 1: Dependability management systems

[4] IEC 60300-2, Dependability management — Part 2: Guidelines for dependability management

[5] KLINE, M.B. Software and hardware R&M: What are the differences? Proceedings of the Annual Reliability and Maintainability Symposium, 1980

[6] LIPOW, M., SHOOMAN, M. L. Software reliability, Tutorial Session, Annual Reliability and Maintainability Symposium, 1986

[7] ISO/IEC 15288, Systems and software engineering — System life cycle processes

[8] Capability Maturity Model® (CMM®), Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA USA

[9] Capability Maturity Model Integration® (CMMI®) for Development, Version 1.2; Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA USA 2006

[10] IEC 60300-3-3, Dependability management — Part 3-3: Application guide — Life cycle costing

[11] IEC 62347, Guidance on system dependability specifications

[12] IEC 61160, Design review

[13] CHILLAREGE, Ram. Orthogonal Defect Classification — A concept for in process Measurements, IEEE Transactions on Software Engineering, 1992

[14] LYU, M.R. (Ed.): The Handbook of Software Reliability Engineering, IEEE Computer Society Press and McGraw-Hill Book Company, 1996

[15] IEEE-1633: Recommended Practice on Software Reliability, 2009

[16] ISO/IEC 20926, Software and systems engineering — Software measurement — IFPUG functional size measurement method 2009

[17] IEC 61078, Analysis techniques for dependability — Reliability block diagram and boolean methods

[18] IEC 61025, Fault tree analysis (FTA)

[19] IEC 61165, Application of Markov techniques

[20] IEC 62551, Analysis techniques for dependability — Petri net techniques2

[21] DUGAN, J.B. Fault Tree Analysis for Computer-based Systems, Tutorial Session, Annual Reliability and Maintainability Symposium, 2000

[22] IEC 62198, Project risk management — Application guidelines

[23] IEC 60812, Analysis techniques for system reliability — Procedure for failure mode and effects analysis (FMEA)

[24] IEC 60300-3-1, Dependability management — Part 3-1: Application guide — Analysis techniques for dependability — Guide on methodology

[25] ISO/IEC 15026-3, Systems and software engineering — System and software assurance — Part 3: System integrity levels

[26] IEC 61508-3, Functional safety of electrical/electronic/programmable electronic safetyrelated systems — Part 3: Software requirements

[27] ISO/IEC 13335-1, Information technology — Security techniques — Management of information and communications technology security — Part 1: Concepts and models for information and communications technology security management (withdrawn)

[28] IEC 62429, Reliability growth — Stress testing for early failures in unique complex systems

[29] IEC 61014, Programmes for reliability growth

[30] IEC 61164, Reliability growth — Statistical test and estimation methods

[31] IEC 62506, Methods for product accelerated testing

[32] ISO/IEC/IEEE 42010, Systems and software engineering — Architectural description

[33] ISO/IEC 18019, Systems and software engineering — Guidelines for the design and preparation of user documentation for application software (withdrawn)

[34] Software Assurance Standard, NASA-STD-8739.8 w/Change 1, May 2005

[35] ISO/IEC 15026-4, Systems and software engineering — System and software assurance — Part 4: Assurance in the life cycle2

[36] ISO/IEC 15026-2, Systems and software engineering — System and software assurance — Part 2: Assurance case

[37] LAKEY, P.B., NEUFELDER, A.M. System and Software Reliability Assurance Notebook, Rome Laboratory, 1996

[38] National Information Assurance (IA) Glossary, (CNSS Instruction No. 4009), National Security Telecommunications and Information Systems Security Committee (NSTISSC), published by the United States federal government (unclassified), June 2006

[39] Software Assurance: An Overview of Current Industry Best Practices, Software Assurance Forum for Excellence in Code, February 2008

[40] ISO/IEC TR 12182, Information technology — Categorization of software

УДК 658.562.012.7:65.012.122:006.354 ОКС 03.120.01

Ключевые слова: надежность в технике, программное обеспечение, программный сбой

Редактор *З.Н. Киселева*
Технический редактор *В.Н. Прусакова*
Корректор *О.В. Лазарева*
Компьютерная верстка *Е.А. Кондрашовой*

Сдано в набор 23.09.2021. Подписано в печать 07.10.2021. Формат 60×84⅛. Гарнитура Ариал.
Усл. печ. л. 6,98. Уч.-изд. л. 6,30. Тираж 40 экз. Зак. 1336.

Подготовлено на основе электронной версии, предоставленной разработчиком стандарта

Издано и отпечатано в ФГБУ «РСТ»
117418 Москва, Нахимовский пр-т, д. 31, к. 2.
www.gostinfo.ru info@gostinfo.ru