ФГБОУ ВПО «Воронежский государственный технический университет»

Д.О. Карпеев С.С. Куликов

ТЕХНОЛОГИЯ ПОСТРОЕНИЯ ЗАЩИЩЕННЫХ РАСПРЕДЕЛЕННЫХ ПРИЛОЖЕНИЙ

Утверждено Редакционно-издательским советом университета в качестве учебного пособия

Воронеж 2015

УДК 004.056.5: 004.42

Карпеев Д. О. Технология построения защищенных распределенных приложений: учеб. пособие [Электронный ресурс]. – Электрон. текстовые, граф. данные (1,26 Мб) / Д. О. Карпеев, С. С. Куликов. – Воронеж: ФГБОУ ВПО «Воронежский государственный технический университет», 2015. – 1 электрон. опт. диск (CDROM). – Систем. требования: ПК 500 и выше; 256 Мб ОЗУ; Windows XP; Adobe Reader; 1024x768; CD-ROM; мышь. – Загл. с экрана.

Учебное пособие посвящено разработке распределенного программного обеспечения с учетом требований по обеспечению функциональной и информационной безопасности на всех этапах жизненного цикла разрабатываемых приложений.

Издание соответствует требованиям Федерального государственного образовательного стандарта высшего профессионального образования по специальности 090303 «Информационная безопасность автоматизированных систем», дисциплине «Технология построения защищенных распределенных приложений».

Ил. 23. Библиогр.: 236 назв.

Рецензенты: Концерн «Созвездие»
(канд. техн. наук, ведущий науч. сотрудник О. В. Поздышева);
д-р техн. наук, проф. А. Г. Остапенко

ВВЕДЕНИЕ

В настоящее время много внимания уделяется технологиям разработки распределенных приложений, охватывающих несколько независимых компьютеров. В течение последних десяти лет было создано большое число технологий и стандартов, использование которых должно было помочь разработчикам в создании распределенных приложений масштаба предприятия. Однако поддержка многих технологий была изначально достаточно трудоемкой и сложной для разработчиков прикладных программ, использовавших классические языки программирования, такие как C/C++.

Одной из задач, стоящих перед разработчиками Microsoft, создающими так называемую общеязыковую инфраструктуру (Common Language Infrastructure, CLI), так же известную как .NET, была наиболее полная поддержка средств разработки распределенных систем. Поэтому в платформе разработки приложений Microsoft .NET Framework имеется встроенная поддержка четырех взаимосвязанных технологий, предназначенных для использования в распределенных системах: очередей сообщений (messaging queues), объектов COM+, объектов .NET Remoting, веб-служб (web services).

Каждая из данных технологий имеет свои достоинства, недостатки и особенности применения при построении распределенных приложений. Сделать осознанный выбор с пользу той или иной технологии при решения конкретных прикладных задач трудно без знакомства со всеми ними, также как и без базовых теоретических знаний о распределенных системах.

1. ВВЕДЕНИЕ В РАСПРЕДЕЛЕНИЕ СИСТЕМЫ

1.1. Понятие распределенной системы

По утверждению известного специалиста в области информатики Э. Таненбаума, не существует общепринятого и в то же время строгого определения распределенной системы. Некоторые остряки утверждают, что распределенной является такая вычислительная система, в которой неисправность компьютера, о существовании которого пользователи ранее даже не подозревали, приводит к остановке всей их работы. Значительная часть распределенных вычислительных систем, к сожалению, удовлетворяют такому определению, однако формально оно относится только к системам с уникальной точкой уязвимости (single point of failure).

Часто при определении распределенной системы во главу угла ставят разделение ее функций между несколькими компьютерами. При таком подходе распределенной является любая вычислительная система, где обработка данных разделена между двумя и более компьютерами. Основываясь на определении Э. Таненбаума, несколько более узко распределенную систему можно определить как набор соединенных каналами связи независимых компьютеров, которые с точки зрения пользователя некоторого программного обеспечения выглядят единым целым.

Такой подход к определению распределенной системы имеет свои недостатки. Например, все используемое в такой распределенной системе программное обеспечение могло бы работать и на одном единственном компьютере, однако с точки зрения приведенного выше определения такая система уже перестанет быть распределенной. Поэтому понятие распределенной системы, вероятно, должно основываться на анализе образующего такую систему программного обеспечения.

Как основу описания взаимодействия двух сущностей рассмотрим общую модель взаимодействия клиент-сервер, в которой одна сторон (клиент) инициирует обмен данными,

посылая запрос другой стороне (серверу). Сервер обрабатывает запрос и при необходимости посылает ответ клиенту (рис. 1.1).

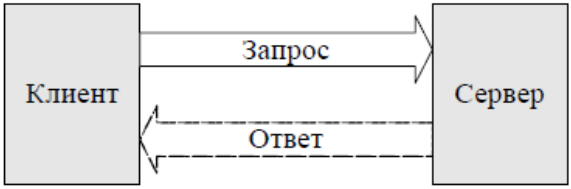

Рис. 1.1. Модель взаимодействия клиент-сервер

Взаимодействие в рамках модели клиент-сервер может быть как синхронным, когда клиент ожидает завершения обработки своего запроса сервером, так и асинхронным, при котором клиент посылает серверу запрос и продолжает свое выполнение без ожидания ответа сервера. Модель клиента и сервера может использоваться как основа описания различных взаимодействий. Для данного курса важно взаимодействие составных частей программного обеспечения, образующего распределенную систему.

Рассмотрим некое типичное приложение, которое в соответствие с современными представлениями может быть разделено на следующие логические уровни (рис. 1.2): пользовательский интерфейс (ИП), логика приложения (ЛП) и доступ к данным (ДД), работающий с базой данных (БД). Пользователь системы взаимодействует с ней через интерфейс пользователя, база данных хранит данные, описывающие предметную область приложения, а уровень логики приложения реализует все алгоритмы, относящиеся к предметной области.

Поскольку на практике разных пользователей системы обычно интересует доступ к одним и тем же данным, наиболее простым разнесением функций такой системы между несколькими компьютерами будет разделение логических уровней приложения между одной серверной частью приложения, отвечающим за доступ к данным, и

находящимися на нескольких компьютерах клиентскими частями, реализующими интерфейс пользователя. Логика приложения может быть отнесена к серверу, клиентам, или разделена между ними (рис. 1.3).

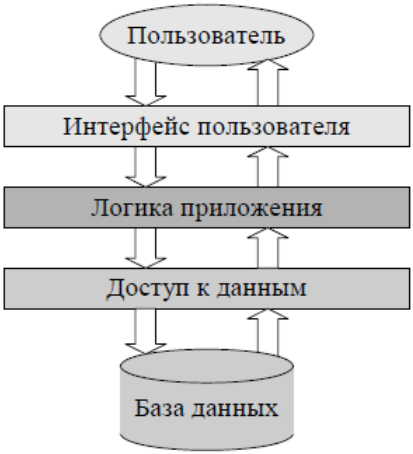

Рис. 1.2. Логические уровни приложения

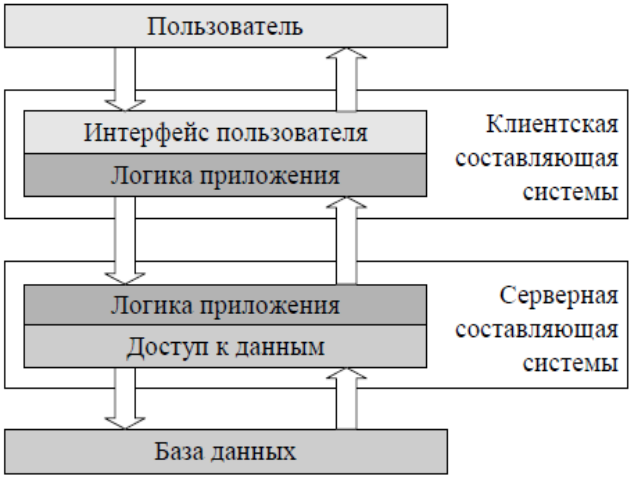

Рис. 1.3. Двухзвенная архитектура

Архитектуру построенных по такому принципу приложений называют клиент-серверной или двухзвенной. На практике подобные системы часто не относят к классу распределенных, но формально они могут считаться простейшими представителями распределенных систем.

Развитием архитектуры клиент-сервер является трехзвенная архитектура, в которой интерфейс пользователя, логика приложения и доступ к данным выделены в самостоятельные составляющие системы, которые могут работать на независимых компьютерах (рис. 1.4).

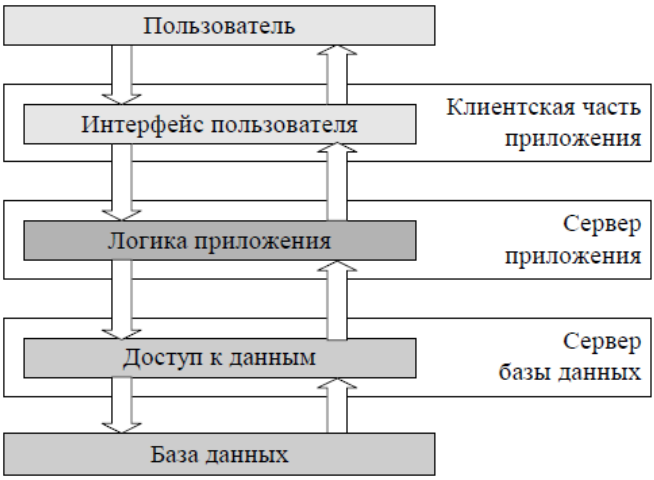

Рис. 1.4. Трехзвенная архитектура

Запрос пользователя в подобных системах последовательно обрабатывается клиентской частью системы, сервером логики приложения и сервером баз данных. Однако обычно под распределенной системой понимают системы с более сложной архитектурой, чем трехзвенная.

Применительно к приложениям автоматизации деятельности предприятия, распределенными обычно называют системы с логикой приложения, распределенной между несколькими компонентами системы, каждая из

которых может выполняться на отдельном компьютере. Например, реализация логики приложения системы розничных продаж должна использовать запросы к логике приложения третьих фирм, таких как поставщики товаров, системы электронных платежей или банки, предоставляющие потребительские кредиты (рис 1.5).

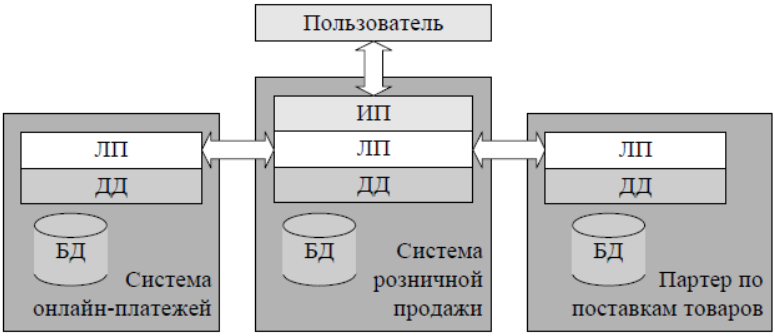

Рис. 1.5. Распределенная система розничных продаж

Таким образом, в обиходе под распределенной системой часто подразумевают рост многозвенной архитектуры «в ширину», когда запросы пользователя не проходят последовательно от интерфейса пользователя до единственного сервера баз данных.

В качестве другого примера распределенной системы можно привести сети прямого обмена данными между клиентами (peer-to-peer networks). Если предыдущий пример имел «древовидную» архитектуру, то сети прямого обмена организованы более сложным образом, рис. 1.6. Подобные системы являются в настоящий момент, вероятно, одними из крупнейших существующих распределенных систем, объединяющие миллионы компьютеров.

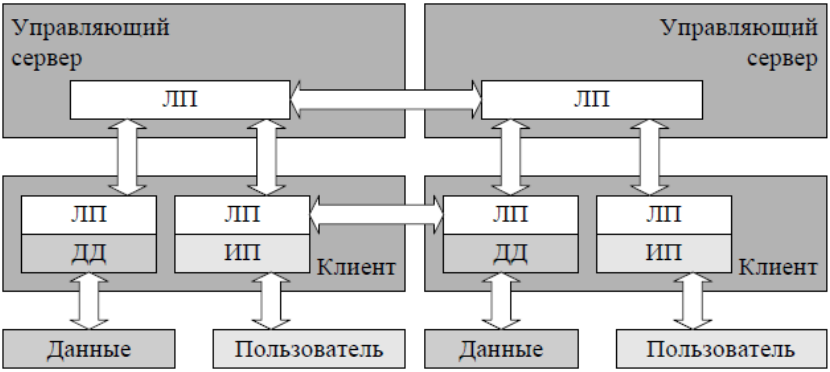

Рис. 1.6. Система прямого обмена данными между клиентами

1.2. Определение распределенной системы. Программные компоненты

В распределенных системах функции одного уровня приложения могут быть разнесены между несколькими компьютерами. С другой стороны, программное обеспечение, установленное на одном компьютере, может отвечать за выполнение функций, относящихся к разным уровням. Поэтому подход к определению распределенной системы, считающей ее совокупностью компьютеров, условен. Для описания и реализации распределенных систем было введено понятие программной компоненты.

Программная компонента – это единица программного обеспечения, исполняемая на одном компьютере в пределах одного процесса, и предоставляющая некоторый набор сервисов, которые используются через ее внешний интерфейс другими компонентами, как выполняющимися на этом же компьютере, так и на удаленных компьютерах. Ряд компонент пользовательского интерфейса предоставляют свой сервис конечному пользователю.

Применительно к программам с использованием платформы CLI, под процессом в приведенном определении компоненты следует понимать домен приложения (application

domain), который можно рассматривать как аналог процесса в управляемом коде.

Основываясь на определении программной компоненты, можно дать более точное определение распределенной системы. Согласно нему, распределенная система есть набор взаимодействующих программных компонент, выполняющихся на одном или нескольких связанных компьютерах и выглядящих с точки зрения пользователя системы как единое целое (рис. 1.7). Прозрачность является атрибутом распределенной системы. При исправном функционировании системы от конечного пользователя должно быть скрыто, где и как выполняются его запросы.

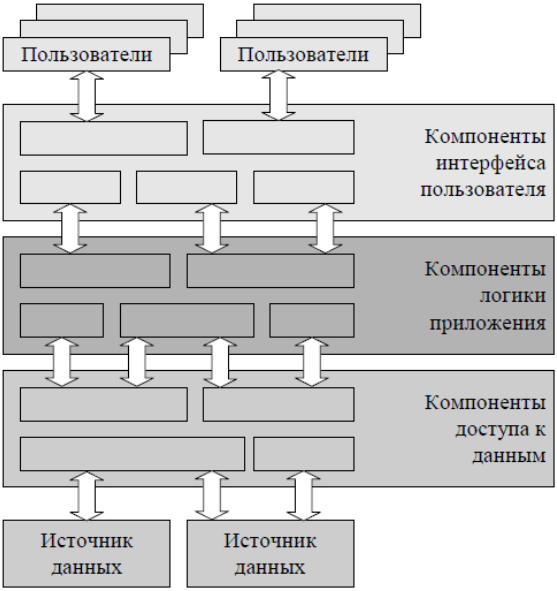

Рис. 1.7. Компоненты распределенной системы

Программная компонента является минимальной единицей развертывания распределенной системы. В ходе модернизации системы одни компоненты могут быть обновлены независимо от прочих компонент.

В хорошо спроектированной системе функции каждой компоненты относятся только к одному уровню приложения. Однако разделение только на три уровня представляется недостаточным для классификации компонент. Например, часть компонент пользовательского интерфейса могут взаимодействовать с пользователем, а часть – предоставлять свои сервисы другим компонентам, но с пользователем не взаимодействовать. Классификации подобного рода существуют, однако они не являются общепринятыми и часто в значительной степени привязаны к приложениям автоматизации деятельности предприятия, что все-таки не является синонимом распределенной системы.

1.3. Требования к распределенным системам

Чтобы достигнуть цели своего существования – улучшения выполнения запросов пользователя – распределенная система должна удовлетворять некоторым необходимым требованиям. Можно сформулировать следующий набор требований, которым в наилучшем случае должна удовлетворять распределенная вычислительная система.

Открытость. Все протоколы взаимодействия компонент внутри распределенной системы в идеальном случае должны быть основаны на общедоступных стандартах. Это позволяет использовать для создания компонент различные средства разработки и операционные системы. Каждая компонента должна иметь точную и полную спецификацию своих сервисов. В этом случае компоненты распределенной системы могут быть созданы независимыми разработчиками. При нарушении этого требования может исчезнуть возможность создания распределенной системы, охватывающей несколько независимых организаций.

Масштабируемость. Масштабируемость вычислительных систем имеет несколько аспектов. Наиболее важный из них для данного курса – возможность добавление в

распределенную систему новых компьютеров для увеличения производительности системы, что связано с понятием балансировки нагрузки (load balancing) на серверы системы. К масштабированию относятся так же вопросы эффективного распределение ресурсов сервера, обслуживающего запросы клиентов.

Поддержание логической целостности данных. Запрос пользователя в распределенной системе должен либо корректно выполняться целиком, либо не выполняться вообще. Ситуация, когда часть компонент системы корректно обработали поступивший запрос, а часть – нет, является наихудшей.

Устойчивость. Под устойчивостью понимается возможность дублирования несколькими компьютерами одних и тех же функций или же возможность автоматического распределения функций внутри системы в случае выхода из строя одного из компьютеров. В идеальном случае это означает полное отсутствие уникальной точки сбоя, то есть выход из строя одного любого компьютера не приводит к невозможности обслужить запрос пользователя.

Безопасность. Каждый компонент, образующий распределенную систему, должен быть уверен, что его функции используются авторизированными на это компонентами или пользователями. Данные, передаваемые между компонентами, должны быть защищены как от искажения, так и от просмотра третьими сторонами.

Эффективность. В узком смысле применительно к распределенным системам под эффективностью будет пониматься минимизация накладных расходов, связанных с распределенным характером системы. Поскольку эффективность в данном узком смысле может противоречить безопасности, открытости и надежности системы, следует отметить, что требование эффективности в данном контексте является наименее приоритетным. Например, на поддержку логической целостности данных в распределенной системе могут тратиться значительные ресурсы времени и памяти,

однако система с недостоверными данными вряд ли нужна пользователям. Желательным свойством промежуточной среды является возможность организации эффективного обмена данными, если взаимодействующие программные компоненты находятся на одного компьютере. Эффективная промежуточная среда должна иметь возможность организации их взаимодействия без затрагивания стека TCP/IP. Для этого могут использоваться системные сокеты (unix sockets) в POSIX-системах или именованные каналы (named pipes).

Устойчивость распределенной системы связана с понятием масштабируемости, но не эквивалентна ему. Допустим, система использует набор обрабатывающих запросы серверов и один диспетчер запросов, который распределяет запросы пользователей между серверами. Такая система может считаться достаточно хорошо масштабируемой, однако диспетчер является уязвимой точкой такой системы. С другой стороны, система с единственным сервером может быть устойчива, если существует механизм его автоматической замены в случае выхода его из строя, однако она вряд ли относится к классу хорошо масштабируемых систем. На практике достаточно часто встречаются распределенные системы, не удовлетворяющие данным требованиям: например, любая система с уникальным сервером БД, реализованным в виде единственного компьютера, имеет уникальную точку сбоя. Выполнение требований устойчивости и масштабируемости обычно связано с некоторыми дополнительным расходами, что на практике может быть не всегда целесообразно. Однако используемые при построении распределенных систем технологии должны допускать принципиальную возможность создания устойчивых и высоко масштабируемых систем.

Классическим примером системы, в значительной мере отвечающей всем представленным выше требованиям, является система преобразования символьных имен в сетевые IP-адреса (DNS). **Система имен** – организованная иерархически

распределенная система, с дублированием всех функций между двумя и более серверами (рис. 1.8).

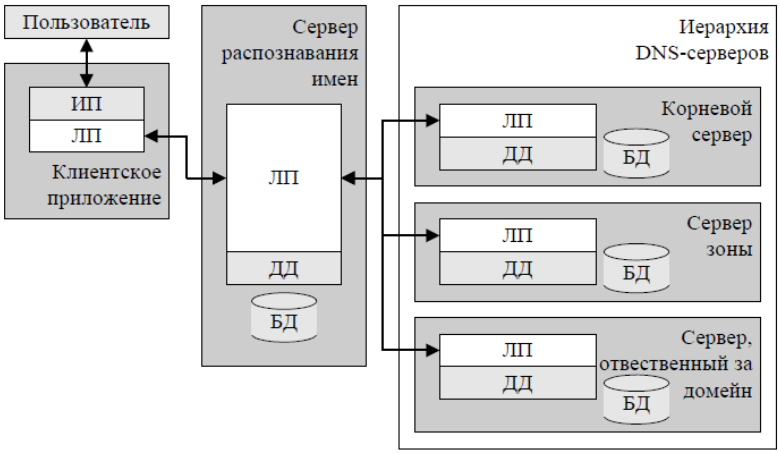

Рис. 1.8. Система DNS

Запрос пользователя на преобразование имени (например, w3c.org) в сетевой адрес передается серверу распознавания имен поставщика услуг интернета. Сервер распознавания имен по очереди опрашивает серверы из иерархии службы имен. Опрос начинается с корневых серверов, который возвращает адреса серверов, ответственных за зону домена. Затем опрашивается сервер, ответственный за зону (в данном случае – .org), возвращающий адреса серверов, ответственных за домен второго уровня, и так далее. Серверы имен кешируют информацию о соответствии имен и адресов для уменьшения нагрузки на систему. Программное обеспечение на компьютере пользователя обычно имеет возможность соединиться с как минимум двумя различными серверами распознавания имен.

Тем не менее, и в системе распознавания имен не все требования распределенным системам выполнены. В частности, она не содержит каких-либо явных механизмов обеспечения безопасности. Это приводит к регулярным атакам

на серверы имен в надежде вывести их из строя, например, большим количеством запросов.

1.4. Понятие промежуточной среды

С точки зрения одного из компьютеров распределенной системы, все другие входящие в нее машины являются удаленными вычислительными системами. Теоретической основой сетевого взаимодействия удаленных систем является общеизвестная модель взаимодействия открытых систем OSI/ISO, которая разделяет процесс взаимодействия двух сторон на семь уровней: физический, канальный, сетевой, транспортный, сеансовый, прикладной, представительский.

В сетях наиболее распространенного стека протоколов TCP/IP протокол TCP является протоколом транспортного, а протокол IP протоколом сетевого уровня. Обеспечение интерфейса к транспортному уровню в настоящее время берет на себя сетевая компонента операционной системы, предоставляя обычно основанный на сокетах интерфейс для верхних уровней. Сокеты обеспечивают примитивы низкого уровня для непосредственного обмена потоком байт между двумя процессами. Стандартного представительского или сеансового уровня в стеке протоколов TCP/IP нет, иногда к ним относят защищенные протоколы SSL/TLS.

Использование протокола TCP/IP посредством сокетов предоставляет стандартный, межплатформенный, но низкоуровневый сервис для обмена данными между компонентами. Для выполнения сформулированных выше требований к распределенным системам функции сеансового и представительского уровня должна взять на себя некоторая промежуточная среда (middleware), называемая так же промежуточным программным обеспечением (рис. 1.9).

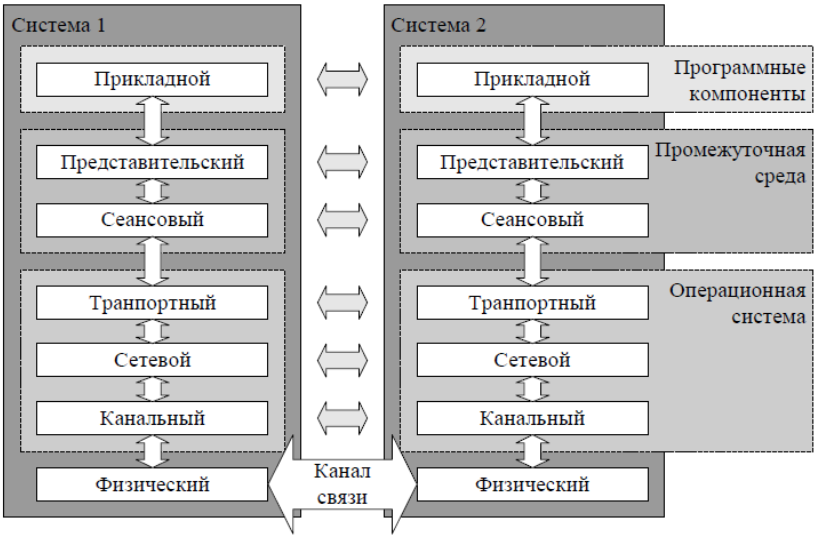

Рис. 1.9. Модель взаимодействия вычислительных систем

Такая среда должна помочь создать разработчиками открытые, масштабируемые и устойчивые распределенные системы. Для достижения этой цели промежуточная среда должна обеспечить сервисы для взаимодействия компонент распределенной системы. К таким сервисам относятся:

- обеспечение единого и независимого от операционной системы механизма использования одними программными компонентами сервисов других компонент;
- обеспечение безопасности распределенной системы: аутентификация и авторизация всех пользователей сервисов компоненты и защита передаваемой между компонентами информации от искажения и чтения третьими сторонами;
- обеспечение целостности данных: управление транзакциями, распределенными между удаленными компонентами системами;

- балансировка нагрузки на серверы с программными компонентами;
- обнаружение удаленных компонент.

В пределах одной распределенной системы может использоваться несколько видов промежуточных сред (рис. 1.10). При хорошем подходе к проектированию системы каждая распределенная ее компонента предоставляет свои сервисы посредством единственной промежуточной среды, и использует сервисы других компонент посредством так же единственной промежуточной среды, однако эти среды могут быть различными.

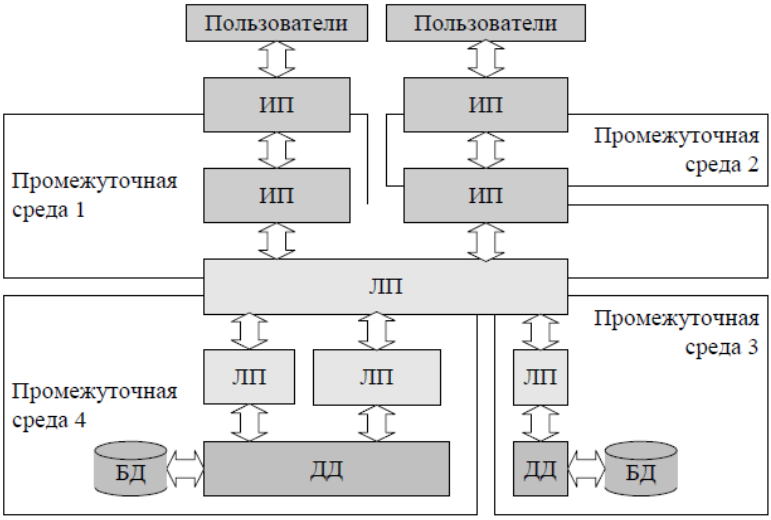

Рис. 1.10. Гетерогенная распределенная система

Распределенную систему, компоненты которой используют несколько промежуточных сред, можно называть гетерогенной, в противоположность гомогенной, использующей единственную промежуточную среду. Поскольку одна и та же промежуточная среда может быть реализована на различных аппаратных платформах и операционных системах, то оба класса распределенных систем

могут включать в себя компьютеры под управлением как одной, так и различных операционных систем.

В настоящий момент нет универсально применимой промежуточной среды, хотя как будет показано в курсе, есть определенное движение в этом направлении. Основной причиной отсутствия такой среды являются отчасти противоречивые требования к распределенным системам, а также различающийся характер сетевых соединений между компонентами системы: например, взаимодействие компонент внутри одного предприятия, вероятно, может быть построено иначе, чем взаимодействие компонент двух различных предприятий, не полностью доверяющих друг другу.

Взаимодействие программных компонент в пределах одного и того же компьютера также происходит при помощи промежуточной среды, что при использовании некоторых промежуточных сред может быть, как неудобно, так и неэффективно. В идеальном случае распределенная компонента должна быть реализована таким образом, чтобы переход с одной промежуточной среды на другую происходил путем изменения конфигурации программной компоненты, а не изменения исходного кода. На практике данное требование, к сожалению, может быть трудно осуществимо, однако необходимо хотя бы минимизировать возможные исправления программного кода при возможной смене промежуточной среды.

2. ВЗАИМОДЕЙСТВИЕ КОМПОНЕНТ РАСПРЕДЕЛЕННОЙ СИСТЕМЫ

2.1. Модели взаимодействия компонент распределенной системы

Ключевым сервисом промежуточной среды для создания распределенных систем является обеспечение обмена данными между компонентами распределенной системы. В настоящий момент существуют две концепции взаимодействия программных компонент: обмен сообщениями между компонентами и вызов процедур или методов объекта удаленной компоненты по аналогии с локальным вызовом процедуры.

Поскольку в настоящее время любое взаимодействие между удаленными компонентами в конечном итоге основано на сокетах TCP/IP, первичным с точки зрения промежуточной среды является низкоуровневый обмен сообщениями на основе сетевых сокетов, сервис которых никак не определяет формат передаваемого сообщения. На базе протоколов TCP или HTTP затем могут быть построены прикладные протоколы обмена сообщений более высокого уровня абстракции для реализации более сложного обмена сообщениями или удаленного вызова процедур.

Удаленный вызов является моделью, происходящей от языков программирования высокого уровня, а не от реализации интерфейса транспортного уровня сетевых протоколов. Поэтому протоколы удаленного вызова должны обязательно базироваться на какой-либо системе передачи сообщений, включая как непосредственное использование сокетов TCP/IP, так и основанные на нем другие промежуточные среды для обмена сообщениями. Реализация высокоуровневых служб обмена сообщениями, в свою очередь, может использовать удаленный вызов процедур, основанный на более низкоуровневой передаче сообщений, использующей, например, непосредственно сетевые сокеты. Таким образом, одна промежуточная среда может использовать для своего

функционирования сервисы другой промежуточной среды, аналогично тому, как один протокол транспортного или сетевого уровня может работать поверх другого протокола при туннелировании протоколов.

2.2. Обмен сообщениями

Существует два метода передачи сообщений от одной удаленной системы к другой – непосредственный обмен сообщениями и использование очередей сообщений. В первом случае передача происходит напрямую, и она возможна только в том случае, если принимающая сторона готова принять сообщение в этот же момент времени. Во втором случае используется посредник – менеджер очередей сообщений. Компонента посылает сообщение в одну из очередей менеджера, после чего она может продолжить свою работу. В дальнейшем получающая сторона извлечет сообщение из очереди менеджера и приступит к его обработке.

Простейшей реализацией непосредственного обмена сообщениями является использование транспортного уровня сети через интерфейс сокетов, минуя какое-либо промежуточное программное обеспечение. Однако такой способ взаимодействия обычно не применяется в системах автоматизации предприятия, поскольку в этом случае реализация всех функций промежуточной среды ложится на разработчиков приложения. При таком подходе сложно получить расширяемую и надежную распределенную систему, поэтому для разработки прикладных распределенных систем обычно используются системы очередей сообщений.

Существует несколько разработок в области промежуточного программного обеспечения, реализующие высокоуровневые сервисы для обмена сообщениями между программными компонентами. К ним относятся, в частности, Microsoft Message Queuing, IBM MQSeries и Sun Java System Message Queue.

Такие системы дают возможность приложениям использовать следующие базовые примитивы по использованию очередей:

- добавить сообщение в очередь;
- взять первое сообщение из очереди, процесс блокируется до появления в очереди хотя бы одного сообщения;
- проверить очередь на наличие сообщений;
- установить обработчик, вызываемый при появлении сообщений в очереди.

Менеджер очереди сообщений в таких системах может находиться на компьютере, отличном от компьютеров с участвующими в обмене компонентами. В этом случае сообщение первоначально помещается в исходящую очередь на компьютере с посылающей сообщения компонентой, а затем пересылается менеджеру требуемой. Для создания крупных систем обмена сообщениями может использоваться маршрутизация сообщений, при которой сообщения не передаются напрямую менеджеру, поддерживающему очередь, а проходят через ряд промежуточных менеджеров очередей сообщений (рис. 2.1).

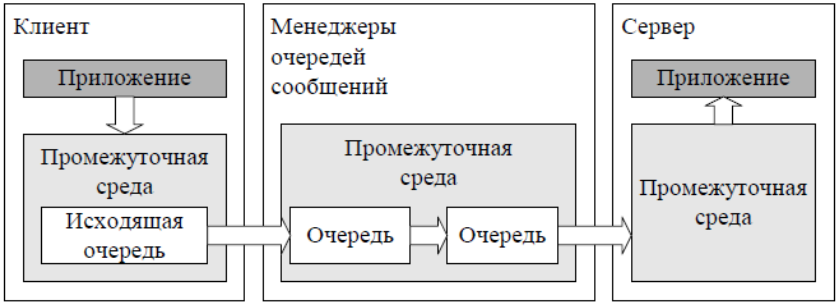

Рис. 2.1. Системы очередей сообщений

Использование очередей сообщений ориентировано на асинхронный обмен данными. Основные достоинства таких систем:

- время функционирования сервера может быть не связано со временем работы клиентов;
- независимость промежуточной среды от средства разработки компонент и используемого языка программирования;
- считывать и обрабатывать заявки из очереди могут несколько независимых компонент, что дает возможность достаточно просто создавать устойчивые и масштабируемые системы.

Недостатки систем очередей сообщений являются продолжением их достоинств:

- необходимость явного использования очередей распределенным приложением;
- сложность реализации синхронного обмена;
- определенные накладные расходы на использование менеджеров очередей;
- сложность получение ответа: передача ответа может потребовать отдельной очереди на каждый компонент, посылающий заявки.

2.3. Дальний вызов процедур

Идея удаленного вызова процедур (remote procedure call, RPC) появилась в середине 80-х годов и заключалась в том, что при помощи промежуточного программного обеспечения функцию на удаленном компьютере можно вызывать так же, как и функцию на локальном компьютере. Чтобы удаленный вызов происходил прозрачно с точки зрения вызывающего приложения, промежуточная среда должна предоставить процедуру-заглушку (stub), которая будет вызываться клиентским приложением. После вызова процедуры-заглушки промежуточная среда преобразует переданные ей аргументы в вид, пригодный для передачи по транспортному протоколу, и передает их на удаленный компьютер с вызываемой функцией. На удаленном компьютере параметры извлекаются

промежуточной средой из сообщения транспортного уровня и передаются вызываемой функции (рис. 2.2). Аналогичным образом на клиентскую машину передается результат выполнения вызванной функции.

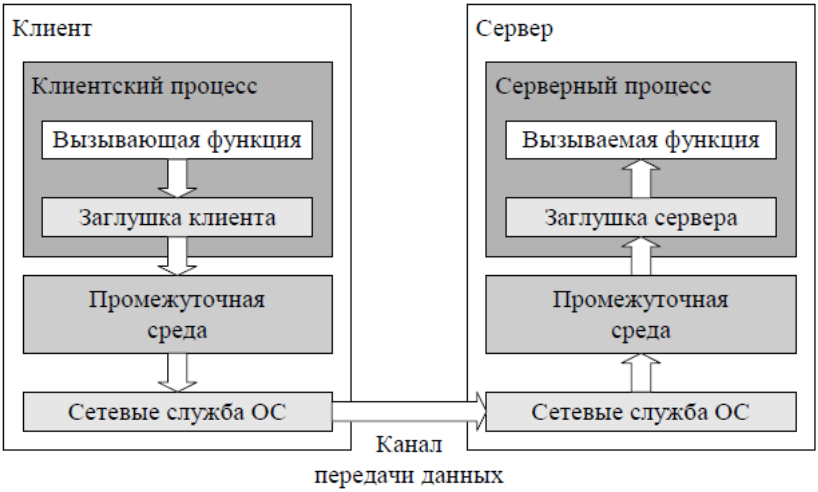

Рис. 2.2. Удаленный вызов процедур

Существует три возможных варианта удаленного вызова процедур:

1. Синхронный вызов: клиент ожидает завершения процедуры сервером и при необходимости получает от него результат выполнения удаленной функции;

2. Однонаправленный асинхронный вызов: клиент продолжает свое выполнение, получение ответа от сервера либо отсутствует, либо его реализация возложена на разработчика (например, через функцию клиента, удалено вызываемую сервером).

Асинхронный вызов: клиент продолжает свое выполнение, при завершении сервером выполнения процедуры он получает уведомление и результат ее выполнения, например через callback-функцию, вызываемую промежуточной средой при получении результата от сервера.

Процесс преобразования параметров для передачи их между процессами (или доменами приложения в случае использования .NET) при удаленном вызове называется маршализацией (marshaling). Преобразование экземпляра какого- либо типа данных в пригодный для передачи за границы вызывающего процесса набор байт называется **сериализацией**. **Десериализация** – процедура, обратная сериализации – заключается в создании копии сериализованного объекта на основе полученного набора байт. Такой подход к передаче объекта между процессами путем создания его копий называется маршализацией по значению (marshal by value), в отличие от рассматриваемой в следующем разделе маршализации по ссылке.

Процесс сериализации должен быть определен для всех типов данных, передаваемых при удаленном вызове. К ним относятся параметры вызываемой функции и возвращаемый функцией результат. В случае передачи параметров по ссылке сериализации подлежат ссылаемые объекты, поскольку для самих указателей сериализация не может быть применена. Это затрудняет использование механизма удаленного вызова в языках, поддерживающих указатели на объекты неизвестного типа.

2.4. Использование удаленных объектов

В связи с переходом разработчиков прикладных программ от структурной парадигмы к объектной появилась необходимость в использовании удаленных объектов (remote method invocation/ RMI). Удаленный объект представляет собой некоторые данные, совокупность которых определяет его состояние. Это состояние можно изменять путем вызова его методов. Обычно возможен прямой доступ к данным удаленного объекта, при этом происходит неявный удаленный вызов, необходимый для передачи значения поля данных объекта между процессами. Методы и поля объекта, которые могут использоваться через удаленные вызовы, доступны через

некоторый внешний интерфейс класса объекта. Внешний интерфейс компоненты распределенной системы в таких системах обычно совпадает с внешним интерфейсом одного из входящих в компоненту классов.

В момент, когда клиент начинает использовать удаленный объект, на стороне клиента создается клиентская заглушка, называемая посредником (proxy). Посредник реализует тот же интерфейс, что и удаленный объект. Вызывающий процесс использует методы посредника, который маршализирует их параметры для передачи по сети, и передает их по сети серверу. Промежуточная среда на стороне сервера десериализует параметры и передает их заглушке на стороне сервера, которую называют каркасом (skeleton) или, как и в удаленном вызове процедур, заглушкой (рис. 2.3). Каркас связывается с некоторым экземпляром удаленного объекта. Это может быть как вновь созданный, так и существующий экземпляр объекта, в зависимости от применяемой модели использования удаленных объектов, которые будут рассмотрены ниже.

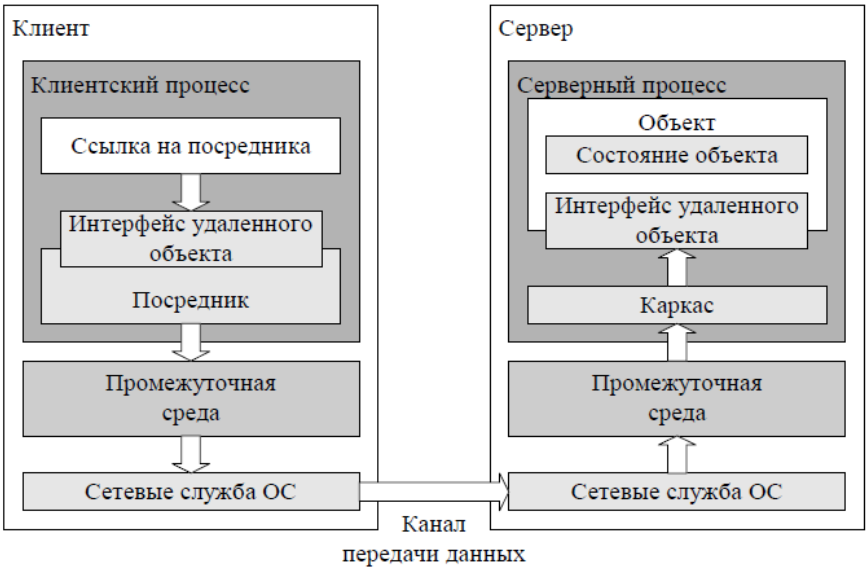

Рис. 2.3. Использование удаленных объектов

Весь описанный процесс называется маршализацией удаленного объекта по ссылке (marshal by reference). В отличие от маршализации по значению, экземпляр объекта находится в процессе сервера и не покидает его, а для доступа к объекту клиенты используют посредников. При маршализации же по значению само значение объекта сериализуется в набор байт для его передачи между процессами, после чего следует создание его копии в другом процессе.

Как и удаленный вызов процедур, вызов метода удаленного объекта может быть как синхронным, так и асинхронным. Существует две особенности при использовании удаленных объектов, не встречавшихся в удаленном вызове процедур. Во-первых, если на момент формирования концепции удаленного вызова процедур исключения (exceptions) еще не поддерживались распространенными языками и не использовались, то в дальнейшем они стали методом информирования вызывающей стороны о встреченных вызываемой стороной проблемах. Таким образом, в системах, использующих удаленные объекты, сериализации подвержены как параметры метода и его результат, так и выбрасываемые при выполнении удаленного метода исключения. Во-вторых, в качестве параметра или результата методов могут тоже передаваться ссылки на удаленный объект (рис 2.4), если вызывающий удаленный метод клиент также может является сервером RMI.

При использовании удаленных объектов проблемными являются вопросы о времени их жизни:

- в какой момент времени создается экземпляр удаленного объекта;
- в течение какого промежутка времени он существует.

Для описания жизненного цикла в системах с удаленными объектами используются два дополнительных понятия:

- активация объекта: процесс перевода созданного объекта в состояние обслуживания удаленного вызова, то есть связывания его с каркасом и посредником.

- деактивация объекта: процесс перевода объекта в неиспользуемое состояние.

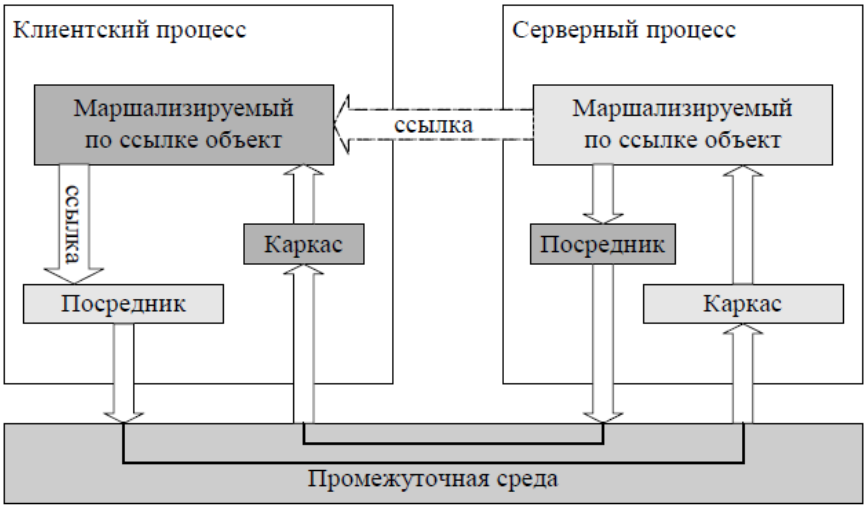

Рис. 2.4. Передача удаленному методу ссылки на объект, маршализируемый по ссылке

Выделяют три модели использования удаленных объектов – модель единственного вызова (singlecall), модель единственного экземпляра (singleton), а также модель активации объектов по запросу клиента (client activation). Первых две модели так же иногда называют моделями серверной активации (server activation), хотя, строго говоря, активация всегда происходит на сервере после какого-либо запроса от клиента.

Модель единственного вызова

При использовании данной модели объект активируется на время единственного удаленного вызова. В наиболее простом случае для каждого вызова удаленного метода объекта клиентом на сервере создается и активируется новый экземпляр объекта, который деактивируется и затем удаляется

сразу после завершения удаленного вызова метода объекта. Таким образом, удаленные вызовы разных клиентов изолированы друг от друга. Благодаря удалению объектов после вызова достигается экономное расходование ресурсов памяти, но могут тратиться значительные ресурсы процессора на постоянное создание и удаление объектов. Посредник на клиенте и заглушка на сервер существуют до уничтожения посредника объекта (рис. 2.5).

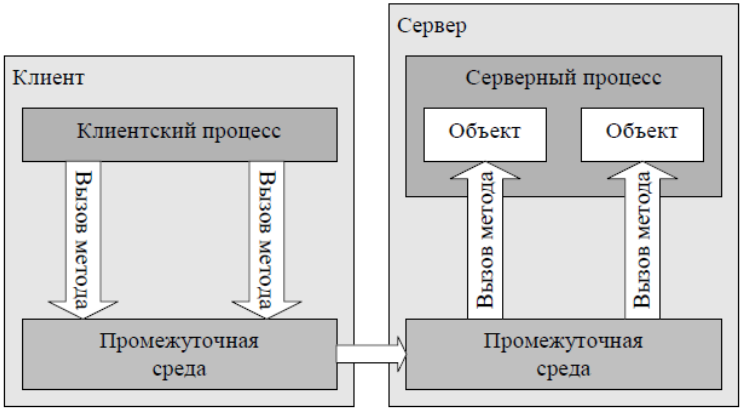

Рис. 2.5. Режим единственного вызова удаленного метода

Данный метод использования удаленных объектов можно рассматривать как некоторый вариант удаленного вызова процедур, поскольку объект не сохраняет свое состояние между вызовами. Тем не менее, сервер использует свои ресурсы на поддержание каркаса и канала между посредником и заглушкой.

Определенным недостатком метода одного вызова является частое создание и удаление экземпляров объектов, поэтому в промежуточным средах может существовать сервис, позволяющий поддерживать некоторое количество уже созданных, но еще не активированных объектов, которые используются для обработки удаленных вызовов. Такой набор

объектов, ожидающих своей активации, называется пулом объектов (*object pooling*). По завершении удаленного вызова объекты деактивируются и могут либо быть помещены в пул и использованы повторно в дальнейшем, либо удаляются, если размер пула достиг некоторого максимального значения. Такая технология позволяет достичь баланса между скоростью обработки запроса и объемом используемых ресурсов сервера. Как видно из описания, в системах с пулом объектов активация не всегда следует непосредственно после создания объекта, а удаление не всегда следует сразу за деактивацией.

Отличительной особенностью метода одного вызова являются наименьшие затраты на организацию системы балансировки нагрузки и ее наибольшая эффективность, поскольку каждый обслуживающий запросы сервер может обработать вызов любого удаленного метода.

Модель единственного экземпляра

При использовании модели единственного экземпляра удаленный объект существует не более чем в одном экземпляре. Созданный объект существует, пока есть хоть один использующий его клиент (рис. 2.6).

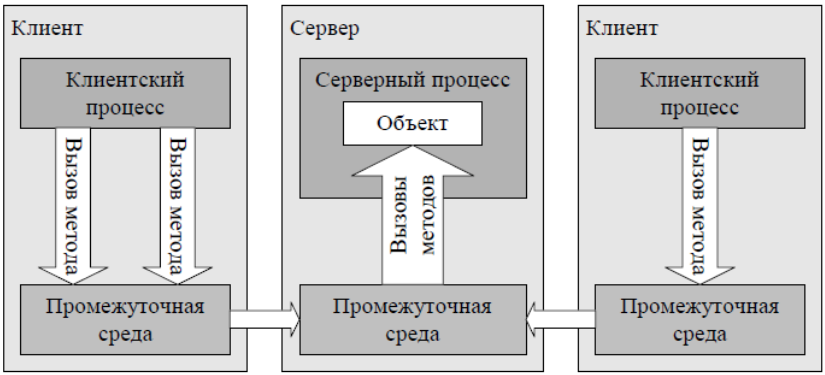

Рис. 2.6. Использование удаленных объектов в режиме единственного экземпляра

При использовании модели единственного объекта вызовы различных клиентов работают с одним и тем же экземпляром удаленного объекта. Поскольку вызовы клиентов не изолированы друг от друга, то используемый объект не должен иметь какого-либо внутреннего состояния. Модель единственного объекта позволяет получить наиболее высокую производительность, поскольку объекты не создаются и не активируются сервером при каждом вызове метода объекта.

Активация по запросу клиента

При каждом создании клиентом ссылки на удаленный объект (точнее, на посредника) на сервере создается новый объект, который существует, пока клиент не удалит ссылку на посредника. При таком методе использования вызовы различных клиентов изолированы друг от друга, и каждый объект сохраняет свое состояние между вызовами, что приводит к наименее рациональному использованию ресурсов памяти сервера (рис. 2.7).

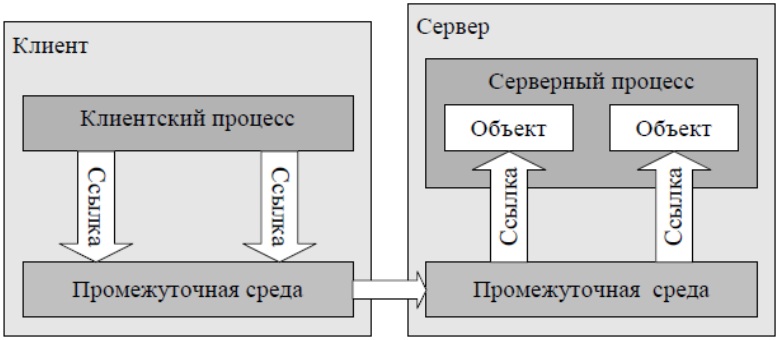

Рис. 2.7. Объекты, активируемые клиентом

Состояние компоненты распределенной системы

Программные компоненты с точки зрения пользователей своих сервисов можно разделить на две категории:

- компоненты без сохраняемого между удаленными вызовами своих методов внутреннего состояния (stateless components);
- компоненты с внутренним состоянием, сохраняемым между удаленными вызовами своих методов (statefull components).

Под состоянием в данном случае понимается совокупность значений полей реализующих компоненту объектов, хранящихся в памяти сервера. Если компонента в ходе своей работы сохраняет какие-либо данные во внешнем хранилище, например в базе данных или очереди сообщений, это обычно не рассматривается как ее внутреннее состояние. Модель единственного вызова не сохраняет состояния удаленного объекта между вызовами его методов, в силу чего данная модель может использоваться только с распределенными компонентами без внутреннего состояния.

Модель одного экземпляра может быть использована для вызова компонент с внутренним состоянием, но это вряд ли часто имеет смысл, поскольку ее состояние будет меняться каждым из клиентов в произвольном порядке. Модель активации по запросу клиента может быть использована с любыми компонентами, но для компонент без внутреннего состояния такой подход обычно ведет к непроизводительному расходу памяти при некотором выигрыше в затратах времени процессора по сравнению с моделью одного вызова.

Компоненты без сохранения внутреннего состояния, используемые вместе с моделью единственного вызова с пулом объектов, имеют наибольшие возможности масштабирования системы при оптимальном балансе между затратами памяти и нагрузкой на процессор.

2.5. Распределенные события

При разработке программного обеспечения достаточно часто возникает потребность получать извещения о каких-либо событиях, возникающих асинхронно, то есть в некоторые произвольные моменты времени. В распределенных системах так же может возникнуть необходимость использования таких извещений, получаемых от удаленной системы. Можно выделить два подхода к обработке событий – тесно связанные и слабо связанные события. При тесно связанном событии происходит прямое уведомление одной стороны другой стороной. Хотя этот метод можно использовать, например, вместе с однонаправленным асинхронным вызовом, ему свойственен ряд недостатков, ограничивающих его применение в распределенных системах:

- обе компоненты системы должны выполняться одновременно;
- для уведомления нескольких компонент об одном событии уведомляющей стороной должны использоваться механизмы для ведения списка получателей событий;
- затруднена фильтрация или протоколирование событий.

Поэтому в распределенных системах так же применяются слабо связанные события, когда источники события (издатели) не взаимодействуют напрямую с получателями событий (подписчиками). Промежуточная среда в этом случае должна предоставить сервис, позволяющий подписчику подписаться на какое-либо событие или отказаться от подписки, а издателю – инициировать событие для рассылки подписчикам (рис. 2.8).

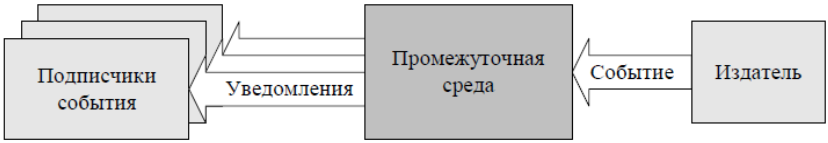

Рис. 2.8. Подписчики и издатели слабосвязанных событий

При использовании слабосвязанных событий подписчики, издатели и менеджер событий могут располагаться на различных компьютерах. Само событие может быть реализовано как, например, вызов менеджером событий некоторого зарегистрированного метода удаленного объекта.

2.6. Распределенные транзакции

Транзакция – последовательность операций с какими-либо данными, которая либо успешно выполняется полностью, либо не выполняется вообще. В случае невозможности успешно выполнить все действия происходит возврат к первоначальным значениям всех измененных в течение транзакции данных (откат транзакции). Транзакция должна обладать следующими качествами.

– **Атомарность**. Транзакция выполняется по принципу «все или ничего».

– **Согласованность**. После успешного завершения или отката транзакции все данные находятся в согласованном состоянии, их логическая целостность не нарушена.

– **Изоляция**. Для объектов вне транзакции не видны промежуточные состояния, которые могут принимать изменяемые в транзакции данные. С точки зрения «внешних» объектов, до успешного завершения транзакции они должны иметь то же состояние, в котором находились до ее начала.

– **Постоянство**. В случае успешности транзакции, сделанные изменения должны иметь постоянный характер (т. е. сохранены в энергонезависимой памяти).

Транзакции являются основой приложений, работающих с базами данных, однако в распределенной системе может быть недостаточно использования только транзакций систем управления базами данных. Например, в распределенной системе в транзакции может участвовать несколько распределенных компонент, работающих с несколькими независимыми базами данных (рис. 2.9).

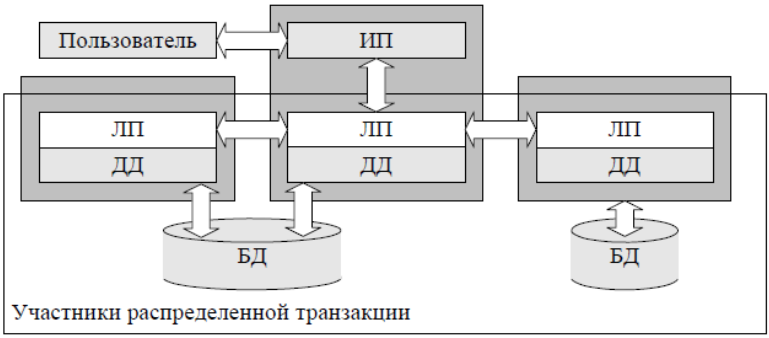

Рис. 2.9. Распределенная транзакция

Распределенной называется транзакция, охватывающая операции нескольких взаимодействующих компонент распределенной системы. Каждая из этих компонент может работать с какими-либо СУБД или иными службами, например, использовать очереди сообщений, или даже работать с файлами. При откате транзакции все эти операции должны быть отменены. Для этого необходимо выполнение двух условий:

- промежуточная среда должна поддерживать управление распределенными между несколькими компонентами транзакциями;
- компоненты распределенной системы не должны работать с какими-либо службами или ресурсами, которые не могут участвовать в транзакции.

Распределенные транзакции являются важнейшим элементом поддержания целостности данных в распределенной системе. Поэтому для более широкого их применения промежуточная среда может содержать механизмы, которые при необходимости (и определенных затратах времени на написание кода) позволят использовать в распределенных транзакциях внешние службы, не поддерживающие транзакции. Такой механизм называется компенсирующим менеджером ресурса (compensating resource manager).

Компенсация в данном случае означает возврат ресурса к первоначальному состоянию при откате транзакции.

В настоящее время происходит формирование и стандартизация еще одного понятия, связанного с поддержкой целостности данных – хозяйственной деятельности (business activity) применительно к распределенным системам. Деятельность обычно является отражением некоторого реального процесса, например, покупки в магазине: от оформления заказа до подтверждения доставки курьером. Деятельность может включать в себя транзакции (оформление заказа покупателя, заказ товара у поставщика, и так далее – до подтверждения доставки покупателем). В отличие от транзакции, время жизни которой предполагается коротким, деятельность может длиться в течение очень долгого времени (например, месяца). Деятельность может поддерживать отмену сделанных изменений (например, оформление возврата товара поставщика при отказе покупателя) путем использования компенсирующих задач.

3. ОБЕСПЕЧЕНИЕ ФУНКЦИОНАЛЬНОЙ БЕЗОПАСНОСТИ РАСПРЕДЕЛЕННЫХ ПРИЛОЖЕНИЙ

3.1. Проблемы обеспечения функциональной безопасности программных средств

Обязательные требования к продукции, производству и эксплуатации определены Федеральным Законом РФ «О техническом регулировании». В нем, в частности, введено понятие безопасности продукции — «состояние, при котором отсутствует недопустимый риск (вероятность), связанный с причинением вреда жизни и здоровью граждан, имуществу физических и юридических лиц, государственному или муниципальному имуществу, окружающей среде...». Там же определены основные понятия: технический регламент, техническое регулирование, стандартизация, сертификация, система сертификации, аккредитация, подтверждение соответствия и др., а также цели, принципы и порядок их практического применения.

В статье из всего многообразия сфер, в которых рассматривается и учитывается безопасность, выделена область применения систем обработки информации и управления реального времени, основой которых являются электронные вычислительные машины (ЭВМ), а также программные средства (ПС) и информационные ресурсы баз данных (БД). При этом под безопасностью продукции, процессов производства и эксплуатации систем понимается их работоспособное состояние и функционирование в соответствии с требованиями заказчика и технической документации, при которых отсутствуют опасные отказы и недопустимый ущерб.

В критических системах, в которых результаты обработки информации и управляющие воздействия непосредственно определяют работоспособность и качество функционирования сложных систем управления в

чрезвычайных ситуациях, системами вооружения, космическими объектами, не исключены и изредка происходят аварии и катастрофы вследствие недостаточной безопасности программных средств. Имеется множество примеров гибели дорогих спутников или катастрофических ситуаций при применении сложных динамических объектов (Марс-1 – 1976 г., Ариан-5 – 1997 г., Апполон-13 – 1978 г. и т. д.), а также больших финансовых и материальных потерь в авиации, в военных системах и на транспорте вследствие относительно простых ошибок в программах. В наиболее тяжелых случаях ущерб измерялся ценой жизни и здоровья людей или большими материальными потерями. В то же время они имели вполне объяснимую природу и источники, влияние которых могло быть снижено путем глубокого анализа, выявления дефектов и корректировок программ или информации баз данных.

Имеющиеся достижения в области теории и практики управления безопасностью сложной промышленной продукции, как правило, не известны и не используются специалистами, создающими и применяющими военные системы на базе программных средств. Это во многих случаях определяет дефекты и отказовые ситуации при применении ПС, конфликты между заказчиками и разработчиками из-за неопределенностей значений их безопасности и не конкурентоспособность создаваемых ПС на мировом рынке. Формализация и оценивание реальной безопасности программных средств и систем часто зависит от интуиции и квалификации их разработчиков, заказчиков и пользователей. В результате проекты ПС не соответствуют исходному, декларированному назначению и первоначальным спецификациям требований к характеристикам безопасности и качества, не укладываются в согласованные графики и бюджет разработки.

В требованиях технических заданий и реализованных проектах сложных систем и ПС, связанных с безопасностью, систематически умалчиваются и/или недостаточно формализуются понятия и метрики безопасности

программного продукта и выдаваемой информации, какими характеристиками они должны описываться, как их следует измерять и сравнивать с требованиями, отраженными в контракте, техническом задании или спецификациях. Кроме того, некоторые из характеристик функциональной безопасности часто вообще отсутствуют в требованиях заказчика и согласованных документах на систему и ПС, что приводит к произвольному их учету или к пропуску при испытаниях. Этому способствует ограниченность ресурсов, необходимых для достижения и оценивания в процессах жизненного цикла комплексов программ, требуемых и реализованных значений безопасности, а также недостаточная формализация и документирование всего процесса их выбора и анализа.

Непрерывно возрастающая сложность и вследствие этого уязвимость систем и ПС от случайных и предумышленных негативных воздействий выдвинули ряд рассматриваемых ниже проблем, связанных с безопасностью систем, использующих программные средства, в разряд важнейших — стратегических, определяющих принципиальную возможность и эффективность их применения. При этом выделились области анализа и обеспечения информационной безопасности, связанные, в основном, с защитой от предумышленных, негативных воздействий на информационные ресурсы систем, и функциональной безопасности, обусловленной отказовыми ситуациями и потерей работоспособности систем и ПС вследствие проявления непредумышленных, случайных дефектов программ, данных, аппаратуры и внешней среды.

Проблема обеспечения информационной безопасности функционирования ИС в процессе разработки и эксплуатации возникла и развивается вследствие возрастания сложности и ответственности задач использования информационных ресурсов и увеличения их уязвимости от предумышленных, внешних воздействий, с целью незаконного использования или искажения информации и программ,

которые по своему содержанию предназначены для применения ограниченным кругом лиц. Основное внимание в современной теории и практике обеспечения безопасности информационных систем сосредоточено на защите от злоумышленных разрушений, искажений, хищений и использования программных средств и информации баз данных. Для этого разработаны и активно развиваются проблемно-ориентированные методы и средства защиты от несанкционированного доступа, от различных типов вирусов и закладок, от утечки информации по каналам электромагнитного излучения и т.д. При этом подразумевается наличие лиц, заинтересованных в несанкционированном доступе к конфиденциальной или полезной информации в системах, с целью её незаконного использования. Для решения этой проблемы созданы и активно развиваются методы, средства и стандарты обеспечения информационной безопасности — защиты программ и данных от предумышленных негативных внешних воздействий.

Проблема обеспечения функциональной безопасности при случайных, дестабилизирующих воздействиях и отсутствии злоумышленного влияния на системы, ПС или информацию баз данных существенно отличается от задач информационной безопасности. При анализе функциональной безопасности рассматриваются опасные отказовые ситуации, приводящие к потере работоспособности систем, к авариям и катастрофам. При таких воздействиях внешняя функциональная работоспособность систем может разрушаться не полностью, однако невозможно полноценное выполнение заданных функций и требований к качеству информации для потребителей. В рассматриваемых ниже системах безопасность их функционирования определяется проявлениями дестабилизирующих факторов, приносящих большой ущерб:

- техническими отказами внешней аппаратуры и искажениями исходной информации от объектов

внешней среды и от потребителей систем и обработанной информации;

- случайными отказами, сбоями и физическими разрушениями элементов и компонентов аппаратных средств вычислительных комплексов и средств телекоммуникации;
- дефектами и ошибками в комплексах программ обработки информации и в данных;
- пробелами и недостатками в средствах обнаружения опасных отказов и оперативного восстановления работоспособного состояния систем, программ и данных.

В реальных сложных системах, связанных с безопасностью, возможны катастрофические последствия и отказы функционирования с большим ущербом при отсутствии враждебных лиц, заинтересованных в подобных нарушениях работоспособности систем и ПС. Вредные и катастрофические последствия таких отказов в ряде областей применения систем могут превышать по результатам последствия злоумышленных воздействий, имеют свою природу, особенности и характеристики. Поэтому они требуют самостоятельного изучения и адекватных методов и средств обеспечения безопасности. В некоторых системах отказы, отражающиеся на функциональной безопасности, могут быть обусловлены нарушением информационной безопасности, предумышленным разрушением или искажением информации в базах данных. Тщательное специфицирование и оценивание функциональной безопасности систем, программного продукта и обработанной для потребителей информации —ключевой фактор обеспечения их эффективного и адекватного применения. Это может быть достигнуто на основе выделения, определения и обеспечения подходящих характеристик с учетом целей использования и функциональных задач ПС и систем.

При анализе характеристик функциональной безопасности целесообразно выделять два класса систем и их ПС. Первый класс составляют системы, имеющие встроенные комплексы программ жесткого регламента реального времени, автоматизировано управляющие внешними объектами или процессами. Время необходимой реакции на отказовые ситуации таких систем обычно исчисляется секундами или долями секунды, и процессы восстановления работоспособности должны проводиться, за это время в достаточной степени автоматизировано (бортовые системы в авиации, в некоторых средствах вооружения и транспорта). Системы второго класса применяются для управления процессами и обработки деловой информации из внешней среды, в которых активно участвуют специалисты-операторы (административные, банковские, штабные военные системы). Допустимое время реакции на опасные отказы в этих системах может составлять минуты, и операции по восстановлению работоспособности могут быть доверены специалистам-администраторам по обеспечению безопасности.

Понятия и характеристики функциональной безопасности систем близки к понятиям надежности. Основное различие состоит в том, что в показателях надежности учитываются все реализации опасных отказов, а в характеристиках функциональной безопасности следует регистрировать и учитывать только те отказы, которые привели к столь большому, катастрофическому ущербу, что отразилось на безопасности функционирования системы, на информации для потребителей или объектов управления. Статистически таких отказов может быть в несколько раз меньше, чем учитываемых в значениях надежности. Однако методы, влияющие факторы и реальные значения показателей надежности ПС могут служить ориентирами при оценке функциональной безопасности критических систем. Поэтому способы оценки и испытаний функциональной безопасности могут базироваться на концепции измерения надежности функционирования комплексов программ и баз данных.

Ущерб от дефектов и ошибок программ и данных может иметь кумулятивный характер и проявляться в систематических отказах, каждый из которых отражается на надежности, но не является катастрофой с большим ущербом, влияющим на безопасность системы. Накопление таких отказов со временем может приводить к последствиям, нарушающим функциональную безопасность систем и их применение. Таким образом, сближаются понятия надежности и функциональной безопасности сложных систем и ПС. При более или менее одинаковых источниках угроз и их проявлениях эти понятия можно разделить по величине последствий и ущерба при возникновении отказовых ситуаций.

Проблемы неопределенностей концепции функциональной безопасности конкретных систем, включающих программные средства, должны учитываться заказчиками, пользователями и разработчиками в течение всего их жизненного цикла. Чем сложнее системы и чем выше к ним требования безопасности, тем неопределеннее функции и характеристики их безопасности и качества. Неопределенности начинаются с требований заказчиков, которые при формулировке технического задания и спецификаций не полностью формализуют и принципиально не могут обеспечить содержание абсолютно всего набора функций, характеристик и их значений безопасности, которые должны быть при завершении проекта и предъявлении конечного продукта заказчику. Эти требования итерационно формируются, детализируются и уточняются по согласованию между всеми участниками проекта вследствие естественной ограниченности первичных исходных данных и изменения их под влиянием объективных и субъективных воздействий со стороны различных процессов на последовательных этапах ЖЦ.

Всегда не полностью, с необходимой детализацией определены и описаны все характеристики, особенности функционирования и безопасности объектов внешней среды. Эти характеристики в той или иной степени обычно находятся

под воздействием управляемой системы и ПС. Сложность, а поэтому и неопределенность их представления, как правило, адекватны сложности всей системы, функциональная безопасность которой должна обеспечиваться в течение ее ЖЦ. Квалификация и субъективные свойства потребителей и пользователей изменяются по мере освоения функциональных возможностей системы и ее работоспособности, что увеличивает неопределенность ее реальной безопасности. Смена и различия персонала, применяющего систему и ПС, дополнительно увеличивает неопределенность значений безопасности и трудности ее прогнозирования с учетом множества субъективных факторов различных специалистов, участвующих в эксплуатации.

В процессе проектирования, разработки и всего жизненного цикла основных функциональных задач, операционной среды, аппаратуры ЭВМ эти компоненты с течением времени развиваются и адаптируются, что отражается на необходимости адекватного изменения методов, задач и средств обеспечения их функциональной безопасности. Таким образом, проблемы обеспечения функциональной безопасности сложных систем должны решаться с учетом одновременного динамического развития всех компонентов среды и факторов, непрерывно изменяющихся и воздействующих на результаты их решения. Однако такой сложный, непрерывный, многосвязный процесс трудно реализовать практически и его целесообразно решать поэтапно, возможно с необходимыми итерациями и упрощениями. При этом следует иметь в виду, что всегда могут проявиться отдаленные связи процессов, которые могут существенно повлиять на текущие работы по обеспечению безопасности системы и ПС.

Роль негативных воздействий и их разрушительные последствия быстро возрастают в связи с ростом сложности разработки и применения современных систем на базе ЭВМ и ответственности решаемых ими задач. Одновременно возрастает сложность внешней и операционной среды, в

которой функционируют ПС и ответственность систем, связанных с безопасностью. Объективное повышение сложности функций, реализуемых программами в современных системах, непосредственно приводит к увеличению их объема и трудоемкости создания. Соответственно росту сложности программ возрастает относительное и абсолютное количество выявляемых и остающихся в них дефектов и ошибок, что отражается на снижении потенциальной безопасности их функционирования. По мере увеличения сложности задач, решаемых программами, возрастает влияние ошибок, которые могут угрожать авариями и катастрофами в системах, выполняющих критические функции управления крупными, дорогими и особенно важными объектами или процессами.

Упорядоченное, регламентированное проектирование архитектуры, разработка и сопровождение сложных ПС на базе современных технологий позволяет предупреждать и устранять наиболее опасные системные, алгоритмические и программные дефекты и ошибки на ранних стадиях жизненного цикла, а также использовать неоднократно проверенные в других проектах безопасные программные и информационные компоненты. Для обеспечения безопасности критических систем необходимы эффективные методы и средства, предупреждающие и выявляющие дефекты, а также удостоверяющие безопасность использования программ и баз данных, оперативно защищающие их корректное функционирование при проявлении любых дефектов и отказовых ситуаций.

Работоспособность ПС может быть обеспечена при исходных данных, которые использовались при их разработке, отладке и испытаниях. Реальные исходные данные могут иметь значения, отличающиеся от предусмотренных техническим заданием и от используемых при эксплуатации программ и баз данных. При таких исходных данных функционирование ПС трудно предсказать заранее, и весьма вероятны различные аномалии, завершающиеся отказами, которые отражаются на безопасности. Следует учитывать принципиальные трудности

аналитического оценивания и прогнозирования значений функциональной безопасности программных средств, вследствие непредсказуемости положения, проявления и последствий дефектов и ошибок в программах и данных. Это приводит к практической невозможности достоверных априорных аналитических расчетов функциональной безопасности комплексов программ при ее высоких значениях.

Проблема достижения требуемой функциональной безопасности систем, содержащих программные средства реального времени, решается путем использования современных регламентированных технологических процессов и инструментальных средств обеспечения их жизненного цикла. Они должны быть поддержаны группой международных стандартов, определяющих состав и процессы выполнения требований к заданной функциональной безопасности систем и ПС. Структура, последовательность и содержание технологических процессов ЖЦ в этих стандартах несколько различаются, однако номенклатура базовых компонентов практически совпадает, что позволяет их выбирать и применять с учетом особенностей обеспечения безопасности конкретных проектов ПС.

Для систематической, координированной борьбы с угрозами безопасности ПС необходимы исследования факторов, влияющих на функциональную безопасность со стороны случайных дефектов и ошибок, существующих и потенциально возможных в конкретных системах и комплексах программ. Это позволит целенаправленно разрабатывать методы и средства обеспечения функциональной безопасности критических ПС различного назначения при реально достижимом снижении уровня дефектов проектирования и разработки. Проблема в значительной степени решается посредством применения современных методов, инструментальных средств и стандартов, поддерживающих системный анализ, технологию проектирования, разработки и сопровождения систем, их программных средств и баз данных.

Для создания безопасных систем и ПС, прежде всего, следует формализовать их назначение, функции и основные характеристики. На этой основе должны разрабатываться общие требования к функциональной безопасности и другим характеристикам качества ПС, к обработанной информации для потребителей, адекватной назначениям и функциям систем. Требования к функциям систем и ПС, а также к безопасности их функционирования должны соответствовать доступным ресурсам для их реализации с учетом допустимого ущерба — рисков вследствие отказов при неполном выполнении требований. Ограниченности ресурсов различных видов для обеспечения функциональной безопасности значительно влияют на технико-экономические показатели, качество и функциональную безопасность всей системы и ПС. В результате сложность программ и баз данных, а также доступные ресурсы для их реализации становятся косвенными критериями или факторами, влияющими на выбор методов разработки, на достигаемое качество и безопасность ПС.

Для обеспечения безопасности ПС и результирующей информации для потребителей, необходимо освоение и применение современных методов, автоматизированных технологий и инструментальных средств, обеспечивающих предотвращение или исключение большинства видов дефектов и ошибок при создании и модификации ПС и их компонентов, обеспечивающих безопасность. Все этапы разработки и сопровождения ПС следует поддерживать методами и средствами верификации и систематического, автоматизированного тестирования модулей и компонентов программ. Тестирование является основным методом устранения дефектов, измерения и определения реальных характеристик программ на любых этапах их жизненного цикла. Наличие достаточно полных эталонов на основе совокупности требований спецификаций и поэтапная их декомпозиция — необходимая база тестирования и измерения реальной безопасности и качества комплексов программ. Ограниченность ресурсов при создании ПС приводит к

целесообразности тщательного планирования, упорядочения и применения экономичных и эффективных методов автоматизации поэтапных испытаний в ЖЦ ПС с целью достижения требуемой функциональной безопасности и достоверного ее определения.

Разработку систем и ПС должны завершать комплексные испытания и удостоверение достигнутой функциональной безопасности и надежности систем с программными средствами, предусматривающие возможность совершенствования их характеристик путем соответствующих корректировок программ. Повышение функциональной безопасности целесообразно путем реализации процедур анализа выявленных дефектов и оперативного восстановления вычислительного процесса, программ и данных (рестарта) после обнаружения аномалий и отказов функционирования ПС. Этому может способствовать накопление, мониторинг и хранение данных о выявленных дефектах, сбоях и отказах в процессе исполнения программ и обработки данных.

3.2. Основные понятия и факторы, определяющие безопасность программных средств

Любые программные средства, прежде всего, должны иметь экономическую, техническую, научную или социальную эффективность применения, которая в проектах должна отражать основную цель их жизненного цикла в системе. Эта системная эффективность может быть описана количественно или качественно, в виде набора полезных свойств и характеристик ПС, их отличий от имеющихся у других комплексов программ, а также факторов и источников возможной эффективности. В результате должна быть формализована цель использования и набор требований заказчика и пользователей при создании или приобретении ПС, а также его предполагаемое назначение и сфера применения.

Системная эффективность применения программных средств определяется степенью удовлетворения потребностей определенных лиц — заказчиков и/или пользователей, которую во многих случаях желательно измерять экономическими категориями: прибылью, стоимостью, трудоемкостью, предотвращенным ущербом, длительностью применения и т.п. В стандартах эта эффективность отражается основной, обобщенной характеристикой качества — функциональная пригодность ПС . Данная характеристика связана с тем, какие функции и задачи решает ПС для удовлетворения потребностей пользователей, в то время как другие конструктивные характеристики (в том числе безопасность) главным образом связаны с тем, как и при каких условиях заданные функции могут выполняться с требуемым качеством. Номенклатура и значения всех конструктивных показателей качества непосредственно определяются требуемыми функциями программного средства и, в той или иной степени, влияют на выполнение этих функций. Поэтому выбор и формализация функциональной пригодности ПС, подробное и корректное описание ее свойств являются основными исходными данными для установления при проектировании требуемых значений всех остальных стандартизированных показателей качества.

Стандартизированные в ISO 9126 характеристики качества ПС различаются по степени влияния на системную эффективность применения комплексов программ по прямому назначению. Высшие приоритеты, естественно, должны присваиваться свойствам и атрибутам функциональной пригодности, необходимым для достижения основных стратегических целей использования ПС. Все остальные стандартизированные характеристики ПС должны способствовать обеспечению тактических целей выбранных конструктивных требований. Решение этих задач должно быть направлено на обеспечение высокой функциональной пригодности ПС путем сбалансированного улучшения остальных конструктивных характеристик качества (и

безопасности) в условиях ограниченных ресурсов на ЖЦ. Для этого в процессе системного анализа при подготовке технического задания и требований спецификаций, значения различных факторов, характеристик качества и безопасности должны выбираться с учетом их влияния на функциональную пригодность. На рис. 3.1. представлена основная совокупность факторов и действий по обеспечению функциональной безопасности систем и ПС.

Улучшение каждой вспомогательной — конструктивной характеристики качества, в том числе безопасности, требует некоторых затрат ресурсов (трудоемкости, финансов, времени), которые в той или иной степени должны отражаться на основной характеристике качества — на функциональной пригодности. В зависимости от назначения и функций ПС почти каждая из конструктивных характеристик может стать доминирующей или даже почти полностью определяющей функциональную пригодность ПС. В наибольшей степени функциональная пригодность во многих случаях зависит от безопасности и надежности ПС.

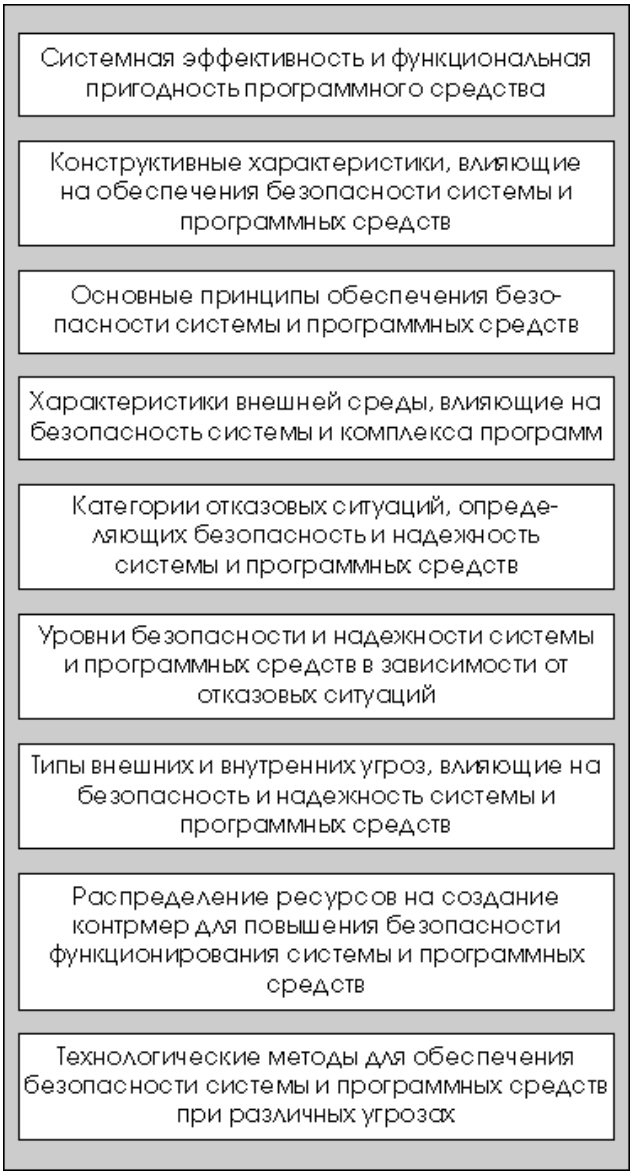

Рис. 3.1. Основная совокупность факторов и действий по обеспечению функциональной безопасности систем и ПС

Цели, назначение и функции защиты комплекса программ от отказов тесно связаны с особенностями функциональной пригодности каждого типа ПС. Разработка и формирование требований к свойствам и характеристикам безопасности должны осуществляться на основе потребностей эффективной реализации назначения и основных функций ПС при различных, реальных угрозах. В процессе системного анализа и проектирования должны быть выявлены потенциальные предумышленные и случайные угрозы функционированию ПС и установлен необходимый уровень безопасности данного комплекса программ. В соответствие с этим уровнем, заказчиком и разработчиками должны выбираться и устанавливаться требуемые и необходимые наборы методов, свойств и средств обеспечения безопасности ПС с учетом ограниченных ресурсов на их реализацию. В результате сформированные требования должны обеспечивать равнопрочную защиту от различных реальных угроз и реализацию необходимых мер контроля и подтверждения требуемых характеристик функциональной пригодности комплекса программ в условиях угроз безопасности функционирования ПС.

Для обеспечения эффективности системы, комплекс программ, связанный с безопасностью, целесообразно базировать на следующих общих принципах:

- защита аппаратуры системы, функциональных программ и данных должна быть комплексной и многоуровневой, ориентированной на все виды угроз с учетом их опасности для потребителя;
- стоимость (трудоемкость) создания и эксплуатации системы программной защиты должна быть меньше, чем размеры наиболее вероятного или возможного (в среднем) неприемлемого потребителями системы ущерба — риска от любых потенциальных угроз;
- комплекс программ защиты должен иметь целевые, индивидуальные компоненты контрмер, предназначенные для обеспечения безопасности

функционирования каждого отдельно взятого компонента и функциональной задачи системы с учетом их уязвимости и степени влияния на безопасность системы в целом;

- система программ защиты не должна приводить к ощутимым трудностям, помехам и снижению эффективности применения и решения основных, функциональных задач пользователями системы в целом.

Характеристики внешней среды, прикладные сферы применения комплексов программ, цели и задачи пользователей, уровень автоматизации их функций и многие другие факторы определяют методы и свойства средств обеспечения безопасности вычислительных систем. Различие между видами безопасности не всегда достаточно четкое и его следует рассматривать и учитывать в зависимости от конкретных функций систем и ПС, задач и результатов обеспечения безопасности, а также от категорий и характеристик возможных отказовых ситуаций. При последующем анализе внимание акцентируется на определении требований и применении сложных аппаратно-программных систем и на их функциональной безопасности. Соответственно ущерб при отказовых ситуациях определяется уязвимостью и нарушением корректного выполнения системой основного назначения и требуемых функций при ограниченных ресурсах на их реализацию. Контрмеры при этом ограничиваются дополнительными средствами защиты от отказов, изменением соотношений требований к различным характеристикам ПС и перераспределением доступных ресурсов для их реализации.

Функции систем и их программных средств реализуются в определенной аппаратной, операционной и пользовательской внешней среде, характеристики, которых существенно влияют на функциональную пригодность программ. Для выполнения требуемых функций комплекса программ необходима адекватная исходная информация от объектов внешней среды, содержание которой должно полностью обеспечивать

реализацию функций, декларированных в требованиях к системе. Полнота формализации номенклатуры, структуры и качества входной информации для выполнения требуемых функций, является одной из важных составляющих при определении функциональной пригодности ПС в соответствующей внешней среде.

Требования к функциональной безопасности системы проистекают из целей обеспечения безопасности объектов и их компонентов, реализующих назначение и основные функции системы, а также из целей обеспечения безопасности её среды. Такое разделение основано на совокупном учете инженерного опыта, политики безопасности, экономических факторов и анализа рисков. Требования безопасности являются результатом преобразования общих целей безопасности системы в совокупность требований безопасности для функциональных объектов и требований безопасности для внешней среды, которые, в случае их удовлетворения, должны обеспечить для системы и ПС способность достижения его базовых целей функциональной безопасности.

Так как без дефектов и ошибок принципиально невозможно создать и применять сложные комплексы программ, эти проблемы должны направленно изучаться и обобщаться в некоторую совокупность знаний — «дефектологию ПС». В ней следует оценивать свойства, содержание и характеристики дефектов и ошибок в ПС, их проявления и негативные последствия для потребителя, угрозы и ущерб для безопасности функционирования системы в целом. Внимание должно быть сосредоточено на характеристиках дефектов функциональных программ, определяющих основное назначение системы.

Среда обеспечения функциональной безопасности ПС включает политики и программы организации безопасности предприятий и систем, опыт, специальные навыки и знания, определяющие контекст предполагаемого применения системы. Среда включает также возможные угрозы безопасности, присутствие которых в этой среде установлено

или предполагается. При формализации среды функциональной безопасности следует принимать во внимание:

- предназначение системы и ПС, включая функции продукта и предполагаемую сферу его применения;
- активы — программы и данные функциональных задач системы, которые требуют обеспечения безопасности, к которым применяются требования и/или политики безопасности, а также компоненты, которые подчинены требованиям безопасности системы и ПС;
- физическую среду в той её части, которая определяет все аспекты эксплуатационной среды системы, касающиеся безопасности, включая мероприятия, относящиеся к средствам физической защиты и к персоналу.

На основании разработанных политик безопасности, оценок угроз и рисков следует сформировать исходные данные, относящиеся к функциональной безопасности среды системы и основного комплекса программ:

- предположения, которым должна удовлетворять среда для того, чтобы система или ПС считались безопасными;
- угрозы безопасности для активов, в которых были бы идентифицированы все угрозы среды, прогнозируемые на основе анализа безопасности как относящиеся к объекту функциональной безопасности;
- угрозы, которые раскрываются через понятия источника угроз, предполагаемого метода их реализации, предпосылки для отказов и идентификация компонентов, которые являются объектами отказов;
- политика безопасности, в которой были бы достаточно точно идентифицированы и описаны цели, методы и правила реализации функциональной безопасности системы.

Результаты анализа среды безопасности могут использоваться для формулирования целей функциональной

безопасности системы, которые направлены на противостояние установленным угрозам, а также проистекают из политики безопасности предприятия или проекта и сделанных предположений. Необходимо, чтобы общие цели безопасности системы были согласованы с определенными ранее целями применения и характеристиками функциональной пригодности системы или ПС как продукта, а также со всеми известными сведениями о внешней среде системы.

Основные понятия в области функциональной безопасности систем и программных средств характеризуются факторами, связанными с возникновением опасных состояний, приводящих к потере работоспособности. Если эти события не обнаруживаются и не устраняются специально введенными в состав системы средствами обеспечения безопасности, то возникают опасные отказы и их последствия. Систему, в состав которой требуется вводить функции безопасности, принято называть системой, связанной с безопасностью. Эти функции необходимы для достижения или поддержания безопасного состояния объекта управления или обработки информации — самостоятельно или совместно с другими, связанными с безопасностью системами, а также с внешними средствами снижения риска для предотвращения или смягчения последствий опасного события.

Примерами могут служить ПС систем, связанные с безопасностью управления движением самолета, автомобиля, систем электрической и диспетчерской централизации на железной дороге, средств управления атомными реакторами и т. д. Иными словами, любое программное средство, отказ которого может повлиять на возникновение аварийной или катастрофической ситуации, последствиями которой может быть тот или иной ущерб, следует считать связанным с безопасностью. Человек может являться компонентом такой системы, например, получать информацию от программируемого электронного устройства и выполнять действия по обеспечению безопасности на основе этой

информации или же он может выполнять действия по обеспечению безопасности, используя такое устройство.

Эти определения базируются на понятии отказа как события, заключающегося в нарушении работоспособного состояния объекта. Определение отказа является базовым для оценки функциональной безопасности и надежности технических систем:

- **работоспособное состояние** — при котором значения всех параметров, характеризующих способность выполнять заданные функции, соответствуют требованиям нормативно — технической и/или конструкторской и проектной документации;
- **неработоспособное состояние** — при котором значение хотя бы одного параметра, характеризующего способность выполнять заданные функции, не соответствует требованиям нормативно — технической и/или конструкторской и проектной документации;
- **отказовая ситуация** — скрытый отказ, не обнаруживаемый визуально или штатными методами и средствами контроля и диагностирования, но выявляемый средствами автоматизированного рестарта, а также при проведении технического обслуживания или специальными методами диагностики, который потенциально может превратиться в отказ;
- **отказ** — событие, заключающееся в нарушении работоспособного состояния системы;
- опасный отказ — событие, заключающееся в нарушении работоспособного или защитного состояния с большим ущербом;
- **дефекты аппаратуры, программы или данных** — негативные события, заключающиеся в непреднамеренном отклонении от требований спецификации или документации в процессах их жизненного цикла;
- **ошибки аппаратуры, программ или данных** — случайные, непредсказуемые искажения компонентов

или ПС, проявляющиеся в процессах их анализа или функционирования.

Для оценивания свойств функциональной безопасности и надежности систем и ПС необходимо регистрировать, измерять, обобщать и упорядочивать реальные характеристики отказовых ситуаций и их последствия для безопасности. С этой целью в стандарте рекомендуются категории отказовых ситуаций систем и ПС, в зависимости от серьезности их влияния на качество функционирования программ или данных объекта управления. Необходимый для безопасного функционирования уровень качества ПС рекомендуется определять исходя из опасности отказовых ситуаций и возможного при этом ущерба для программ, системы и потребителя. Стандартом установлена следующая классификация отказовых ситуаций (рис. 3.2):

- **категория А — катастрофическая**: отказовая ситуация, которая препятствует полной работоспособности и функционированию системы или ПС в соответствии с требованиями и способна нанести недопустимый по последствиям и величине ущерб системе или пользователям;
- **категория В — опасная/критическая**: отказовая ситуация, которая приводит к значительному снижению работоспособности, возможностей применения и функционирования системы или к отсутствию способности персонала справиться с неблагоприятными эксплуатационными режимами, при которых возникают:
    - крайне тяжелые ситуации или перегрузки системы, которые могут вызвать неточное или неполное выполнение основных, функциональных задач с большим ущербом;
    - недопустимо большое снижение характеристик функциональной пригодности, реализуемых функций и конструктивных характеристик качества системы или ПС;

- неблагоприятные или потенциально фатальные воздействия системы или ПС для окружающей внешней среды;

- **категория C — существенная**: отказовая ситуация, которая приводит к снижению возможностей применения и функциональной пригодности системы или к сокращению способности персонала справиться с неблагоприятными эксплуатационными режимами, при продолжении которых может возникать, например, большое искажение информационных ресурсов или сокращение функциональных возможностей, перегрузки или условия, вызывающие существенное ухудшение работоспособности системы или персонала;
- **категория D — несущественная:** отказовая ситуация, незначительно уменьшающая безопасность функционирования и применения объекта, но отражающаяся на его надежности или требующая действий персонала, которые осуществимы в пределах их возможностей для восстановления нормальной работоспособности, такая ситуация может включать в себя, например, некоторое увеличение рабочей нагрузки, неудобство применения или задержки информации для персонала;
- **категория E — невлияющая**: отказовая ситуация, которая практически не воздействует на работоспособность, эксплуатационные характеристики и возможности объекта или не увеличивает рабочую нагрузку персонала.

Эти пять категорий отказовых ситуаций отражают различную степень их влияния на качество функционирования систем или ПС, которое целесообразно распределить между понятиями безопасность и надежность в зависимости от возможного ущерба — риска при отказах. Первые две категории ситуаций характеризуются большими значениями ущерба при отказах, и их следует рассматривать с позиции классов безопасности функционирования систем. Две

последние категории практически не влияют на безопасность применения объектов, но отражаются на надежности их функционирования с относительно невысоким ущербом. Такие отказовые ситуации не целесообразно рассматривать с позиции безопасности. К наиболее сложному случаю относится третья — существенная категория отказовых ситуаций. Она требует детального и конкретного анализа функциональных характеристик системы, ПС и отказов, которые могут отражаться не только на надежности, но и в той или иной степени влиять на безопасность функционирования системы.

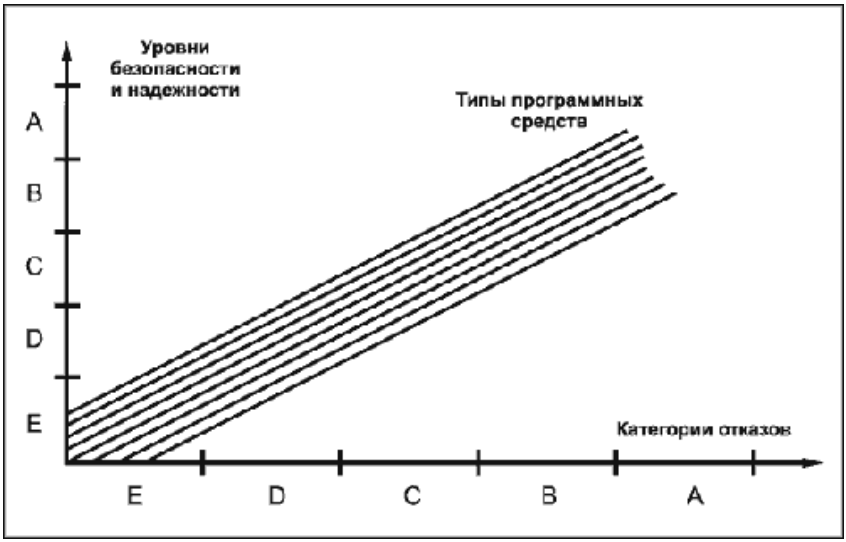

Рис. 3.2. Классификация отказовых ситуаций

Можно предположить для простоты, что уровни функциональной безопасности и надежности линейно зависят от категории отказов и несколько отличаются для различных типов программных средств (рис. 3.2). Эти зависимости можно интерпретировать, как снижение работоспособности системы или ПС в результате проявления некоторого ущерба — риска.

Ущерб при отказах проявляется либо в снижении надежности при функционировании, либо, в худшем случае, как нарушение функциональной безопасности (рис. 3.3). Таким образом, категории отказовых ситуаций, величина ущерба — риска, трудоемкость и длительность восстановления нормального функционирования системы могут рассматриваться как база для оценивания и категоризации пороговых уровней в требованиях к безопасности и/или надежности функционирования и применения систем и их программных средств.

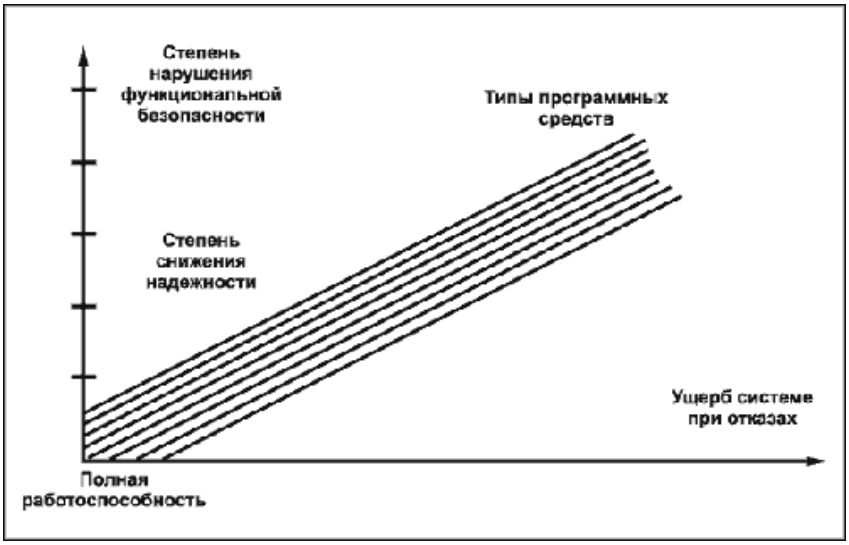

Рис. 3.3. Ущерб при отказах ПС

Уровни безопасности и надежности функционирования систем и ПС стандартом рекомендуется устанавливать в соответствии с категориями наиболее вероятных и опасных отказовых ситуаций в анализируемой или проектируемой системе. Уровень качества ПС также определяется возможностью возникновения потенциальных отказовых ситуаций, в результате сбоев в программах и данных,

выявляемых средствами и процессами оценки безопасности. Это означает, что трудозатраты, ресурсы и время, необходимые для обеспечения согласованности с требованиями заказчика к качеству функционирования, меняются в зависимости от категорий отказовых ситуаций (рис. 3.3):

- **уровень A:** программное средство, аномальное функционирование которого, как показано процессом оценки безопасности и качества системы, может вызвать возникновение отказа ее функционирования, приводящее к катастрофической отказовой ситуации и к полной функциональной непригодности системы с недопустимым ущербом — риском;
- **уровень B**: программное средство, аномальное поведение которого может вызвать возникновение критической отказовой ситуации, опасной для функциональной пригодности системы с большим ущербом — риском;
- **уровень C**: программное средство, аномальное поведение которого при оценке качества системы может способствовать возникновению существенной отказовой ситуации со средним ущербом — риском или значительному снижению функциональной пригодности и надежности;
- **уровень D**: программное средство, аномальное функционирование которого при оценке качества системы способствует возникновению несущественной отказовой ситуации для системы и малому ущербу — риску, влияющему только на некоторое снижение характеристик функциональной пригодности и надежности при применении;
- **уровень E**: программное средство, аномальное поведение которого практически не влияет на работоспособность и основные эксплуатационные возможности системы, работу персонала и риски функционирования, но возможно несколько снижает надежность и качество системы при применении.

Наиболее полно степень функциональной безопасности систем характеризуется величиной предотвращенного или остаточного ущерба — риска, возможного при проявлении дестабилизирующих факторов и реализации конкретных угроз безопасности применения системы и ПС пользователями. С этой позиции затраты ресурсов разработчиками и заказчиками на обеспечение безопасности системы должны быть соизмеримыми с возможным средним ущербом у пользователей от нарушения безопасности. Однако описать и измерить в достаточно общем виде возможный ущерб — риск при нарушении безопасности для критических ПС разных классов весьма сложно. Перечисленные выше отказовые ситуации и уровни функциональной безопасности могут быть следствием различных, негативных явлений в системе, которые в основном оцениваются:

- величиной экономического, материального или социального ущерба системе вследствие возникновения отказовой ситуации;
- трудоемкостью устранения последствий отказовой ситуации и восстановления нормальной работоспособности системы в соответствии с требованиями после отказа;
- длительностью неработоспособного состояния системы вследствие каждого отказа до полного восстановления работоспособности;
- затратами ресурсов и времени, необходимыми после отказа для восстановления нормальной работоспособности системы в соответствии с требованиями.

Реализации угроз, в ряде случаев, целесообразно характеризовать интервалами времени между их проявлениями, нарушающими безопасность применения ПС, или наработкой на отказы, отражающиеся на безопасности. Это сближает понятия и характеристики степени безопасности с показателями надежности системы. Различие состоит в том, что в показателях надежности учитываются все реализации

отказов, а в характеристиках безопасности следует регистрировать только те катастрофические, критические или опасные отказы, которые отразились на нарушении безопасности с большим ущербом. Кроме того, в некоторых случаях последствия отказов может быть полезным отражать длительностью работоспособного состояния системы между событиями отказов относительно общей длительности применения системы с учетом затрат времени на обнаружение и ликвидацию отказов (коэффициент готовности системы).

Значения функциональной безопасности и надежности коррелированы с характеристикой корректность ПС, однако можно достигнуть высокой безопасности функционирования программ при относительно невысокой их корректности за счет автоматизации сокращения ущерба и времени восстановления при отказах. Степень покрытия тестами структуры функциональных компонентов и ПС в целом при разработке и испытаниях может служить ориентиром для прогнозирования их потенциальной безопасности. Распределение реальных длительностей и эффективности восстановления при ограниченных ресурсах для функционирования программ может рассматриваться как дополнительная составляющая при оценивании функциональной безопасности.

В требованиях к каждой конкретной системе должны быть установлены заказчиком совместно с пользователем пороговые значения, разделяющие виды и величины ущерба, а также характеристики восстановления, которые следует относить либо к нарушению функциональной безопасности, либо только к снижению надежности. Однако методы их оценивания и измерения могут быть подобными. Эти пороговые значения зависят от назначения и базовых характеристик, отражающих функциональную пригодность конкретной системы или программного средства. При малом (ниже порога) ущербе вследствие отказовой ситуации и быстром восстановлении такие события следует относить к снижению надежности. Если последствия отказовых ситуаций какой-либо вид ущерба или характеристики восстановления

превышают заданный критический порог, то такие события относятся к нарушению функциональной безопасности (рис. 3.3). При этом в ряде ситуаций для нарушения безопасности может быть достаточным нарушение порогового значения одним из видов отказа. Если аномальное поведение компонента или системы может быть вызвано более чем одной отказовой ситуацией, уровень безопасности ПС следует определять по наиболее серьезной категории отказовой ситуации. Ниже при описании методов испытаний программных средств, для сокращения применяются преимущественно термины «функциональная безопасность» или «безопасность», при этом подразумевается, что так же может оцениваться их надежность.

В основу формирования требований к функциональной безопасности должно быть положено определение перечня и характеристик потенциальных угроз безопасности и установление возможных источников их возникновения. Внешними дестабилизирующими факторами, создающими угрозы безопасности функционированию систем и ПС, являются:

- предумышленные, негативные воздействия лиц с целью искажения, уничтожения или хищения программ, данных и документов системы и ПС, как последствия нарушения информационной безопасности, отражающиеся также на функциональной безопасности;
- ошибки и несанкционированные воздействия оперативного, административного и обслуживающего персонала в процессе эксплуатации системы и ПС;
- искажения в каналах телекоммуникации информации, поступающей от внешних источников и передаваемой потребителям, а также недопустимые значения и изменения характеристик потоков информации от объектов внешней среды;
- сбои и отказы в аппаратуре вычислительных средств;

- вирусы, сбои и отказы, распространяемые по каналам телекоммуникации, влияющие на информационную и функциональную безопасность;
- изменения состава и конфигурации комплекса взаимодействующей аппаратуры системы или ПС за пределы, проверенные при испытаниях или сертификации.

Внутренними источниками угроз безопасности функционирования системы и ПС являются:

- системные ошибки при постановке целей и задач проектирования функциональной пригодности ПС и системы при формулировке требований к функциям и характеристикам средств обеспечения безопасности решения задач;
- дефекты и ошибки при определении функций, условий и параметров внешней среды, в которой предстоит применять защищаемые программные средства и систему;
- алгоритмические ошибки проектирования при непосредственной алгоритмизации функций обеспечения безопасности аппаратуры, программных средств и баз данных при определении структуры и взаимодействия компонентов функциональных комплексов программ, а также при использовании информации баз данных;
- ошибки и дефекты программирования в текстах программ и описаниях данных, а также в исходной и результирующей документации на компоненты ПС;
- недостаточная эффективность используемых методов и средств оперативной защиты программ и данных, обеспечения безопасности функционирования и восстановления работоспособности системы в условиях случайных и предумышленных негативных воздействий от внешней среды.

Полное устранение перечисленных выше угроз безопасности функционирования критических систем

принципиально невозможно. При создании сложных комплексов программ проблема состоит в выявлении факторов, от которых они зависят, в создании методов и средств уменьшения их влияния на безопасность. Необходимо оценивать уязвимость функциональных компонентов системы для различных, негативных воздействий и степень их влияния на основные характеристики качества и безопасности, а также на суммарный риск. В зависимости от этого следует распределять ресурсы для создания системы и ее компонентов, равнопрочных по безопасности функционирования с минимальным обобщенным риском при любых негативных внешних воздействиях. В результате должны быть сформулированы соответствующие методы и контрмеры, которые, в свою очередь, определяют необходимые функции и механизмы средств обеспечения работоспособности и безопасности.

Для обеспечения функциональной безопасности систем создаются соответствующие контрмеры — специализированные системы и средства, которые включают совокупность взаимосвязанных нормативных документов, организационно-технических мероприятий и соответствующих им методов и программных средств, предназначенных для предупреждения и/или ликвидации негативных последствий отказовых ситуаций, различных угроз безопасности, их выявления и локализации. Создание таких комплексов повышения безопасности предусматривает планирование и реализацию целенаправленной политики комплексного обеспечения функциональной безопасности системы, а также эффективное распределение ресурсов на контрмеры и средства обеспечения безопасности. Разработчики и заказчики должны анализировать возможные угрозы, чтобы решать, какие из них действительно присущи внешней среде, системе и ПС. Контрмеры предпринимают для уменьшения уязвимостей и выполнения политики безопасности. Но и после введения этих контрмер могут сохраняться остаточные риски, которые допустимы вследствие ограниченности ресурсов.

Требования к программным средствам, обеспечивающим безопасность, обычно представляются в составе общей спецификации требований к функциональной пригодности системы и ПС. В этом случае должны быть предприняты меры для распределения системных требований к защите между аппаратными и программными компонентами обеспечения функциональной безопасности. Если программная система обеспечения безопасности нуждается во взаимодействии с другими программными или аппаратными компонентами, то в спецификации требований должны быть оговорены непосредственно или при помощи ссылок интерфейсы между применяемыми составляющими.

Для прямых количественных измерений функциональной безопасности необходимы инструментальные средства, встроенные в операционную систему или в соответствующие компоненты и функции ПС. Эти средства должны в динамике реального функционирования ПС автоматически селектировать и регистрировать отказовые ситуации, дефекты и искажения вычислительного процесса программ и данных, выявляемые аппаратным, программно-алгоритмическим контролем или пользователями. Накопление и систематизация проявлений отказов при исполнении программ позволяет оценивать основные показатели безопасности, помогает определять причины сбоев и отказов и подготавливать данные для улучшения функциональной безопасности ПС. Регулярная регистрация и обобщение таких данных способствует устранению ситуаций, негативно влияющих на функциональную пригодность и другие базовые характеристики ПС.

Доступная величина и распределение ресурсов ЭВМ на отдельные виды контрмер оказывают значительное влияние на достигаемую комплексную безопасность системы. Наиболее общим видом ресурсов, который приходится учитывать при проектировании, являются допустимые финансово-экономические затраты или трудоемкость разработки и функционирования комплекса средств обеспечения

безопасности системы. Для размещения этих средств в объектной ЭВМ, при проектировании должна быть предусмотрена программная и информационная избыточности в виде ресурсов внешней и внутренней памяти ЭВМ. Кроме того, для их функционирования необходима временная избыточность — дополнительная производительность ЭВМ. Эти виды вычислительных ресурсов при обеспечении функциональной безопасности используются также для:

- контроля и корректировки искажений информации, поступающей от внешней среды и источников данных;
- оперативного контроля и обнаружения дефектов исполнения программ и обработки данных при использовании ПС по основному назначению;
- размещения и обеспечения функционирования применяемых средств защиты от всех видов угроз безопасности системы;
- генерации тестовых наборов или хранения тестов для контроля работоспособности, сохранности и целостности ПС при функционировании системы;
- накопления, хранения и мониторинга данных о выявленных инцидентах, попытках несанкционированного доступа к информации, о дефектах, сбоях и отказах в процессе исполнения программ и обработки данных, влияющих на безопасность;
- реализации процедур анализа и мониторинга выявленных дефектов и оперативного восстановления вычислительного процесса, программ и данных (рестарта) после обнаружения дефектов и отказов функционирования системы.

Потребителя-заказчика, прежде всего, интересуют функции, безопасность и качество готового конечного продукта — системы и программного средства, и обычно не очень беспокоит, как они достигнуты. Требуемую функциональную безопасность при разработке системы и ПС, как и любой продукции, можно обеспечить двумя методами:

- путем использования только заключительного, выходного контроля и испытаний готовых объектов и исключения из поставки или направлением на доработку продуктов, не соответствующих требуемым безопасности и качеству;
- посредством применения регламентированных технологий и систем обеспечения функциональной безопасности и качества в процессах проектирования, разработки и изготовления, предотвращающих дефекты и гарантирующих высокую безопасность и качество продукции во время ее создания и/или модификации.

Первый метод может приводить к значительным экономическим потерям за счет затрат на создание части не пригодного к использованию брака, что может быть очень дорого для сложных систем. Достижение необходимой безопасности за счет только выходного контроля и испытаний, при отсутствии адекватной технологии и системы обеспечения качества в процессе разработки, может приводить к длительному итерационному процессу массовых доработок и повторных испытаний продукции.

Второй метод обеспечивает высокое качество выполнения всего процесса проектирования, разработки и изготовления и, тем самым, минимум экономических потерь от брака, что более рентабельно при создании сложных систем. При этом сокращается, но не исключается выходной контроль функциональной безопасности и качества продукции. Для создания современных прикладных высококачественных, безопасных систем необходимы оба метода с акцентом на применение регламентированных технологий. Таким образом, политика обеспечения и удостоверения безопасности и качества сложных ПС должно базироваться на проверках и испытаниях:

- технологий обеспечения жизненного цикла программных средств, поддержанных регламентированными системами качества;

- функционирования готового программного продукта с полным комплектом адекватной эксплуатационной документации.

Глубокая взаимосвязь качества разработанных систем и программ с качеством технологии их создания и с затратами на весь жизненный цикл становится особенно существенной при необходимости получения конечного продукта с предельно высокими значениями показателей безопасности и качества. Установлено, что затраты на разработку резко возрастают, когда показатель качества приближается к пределу, достижимому при данной технологии и уровне автоматизации процесса разработки. Это привело к существенному изменению методологии и культуры в области создания и совершенствования систем и ПС. Непрерывный рост требований к функциональной безопасности и качеству ПС стимулировали создание и активное применение международных стандартов и регламентированных технологий, автоматизирующих процессы всего жизненного цикла, начиная с инициирования проекта.

Экспериментальное определение и прямое измерение величины реальной функциональной безопасности сложных систем и комплексов программ в большинстве случаев затруднены необходимостью учитывать и оценивать очень редкие катастрофические отказовые ситуации. Для этого приходится проводить длительные эксперименты с созданием критических и опасных ситуаций эксплуатации систем, что может быть связано с большим риском и весьма дорого. Кроме того, редкие и одиночные отказовые ситуации не позволяют оценить статистически достоверные значения длительностей наработки на отказы и величины вероятного ущерба при этом. Аналитические расчеты и экспертные оценки величины функциональной безопасности комплексов программ усложняют случайное, непредсказуемое размещение дефектов и ошибок в программных модулях и компонентах, которые могут полностью определять функциональную безопасность.

Поэтому высокая функциональная безопасность программных продуктов достигается и определяется преимущественно за счет технологий обеспечения качества на всех этапах жизненного цикла систем и ПС. Для этого в стандартах, регламентирующих функциональную безопасность систем и комплексов программ, значительное внимание уделяется технологическим процессам и инструментальным системам обеспечения безопасности и качества ЖЦ ПС. В стандарте IEC 61508 третья и шестая части полностью посвящены требованиям и регламентированию процессов обеспечения функциональной безопасности встроенных комплексов программ аппаратно-программных систем. Жизненный цикл процессов обеспечения безопасности ПС структурирован схемой этапов и работ, компонентов и объектов, а также снабжен рекомендациями по выбору и реализации методов создания безопасных ПС. Третья часть стандарта ISO 15408 почти целиком посвящена детальным требованиям по обеспечению гарантий качества при создании и применении систем информационной безопасности комплексов. Совокупность этих требований отражает традиционные методы технологии поддержки жизненного цикла сложных ПС с акцентом на особенности реализации функций безопасности, которые применимы и для обеспечения функциональной безопасности. Технологии и типовые процессы создания безопасных систем и программных средств отражены также в стандартах ISO 13335 и IEC 60880. Таким образом, значительная часть требований стандартов, посвященных функциональной безопасности, акцентирована на технологических процессах создания безопасных систем и программных средств, однако без выделения и описания процессов измерения достигаемых при этом критериев безопасности программных продуктов.

3.3. Характеристики среды, для которой должна обеспечиваться функциональная безопасность программных средств

Функциональная безопасность необходима не для всех классов современных систем и прикладных программных средств. Для конкретизации области дальнейшего анализа целесообразно выделить и установить обобщенные характеристики и атрибуты рассматриваемых критических ПС в соответствии со стандартом ISO 12182 — (Классификация программных средств), для которых целесообразно обеспечивать функциональную безопасность. В стандарте выделены три группы видов характеристик: внутренние виды; виды среды и виды данных. Для каждого вида представлен перечень классов, из которых рекомендуется выбирать подходящие характеристики для отражения особенностей конкретной системы или для достаточно широкой сферы применения ПС. Из общего числа трех видов, 16-ти классов и около ста типов характеристик ПС для обеспечения их функциональной безопасности далее будут учитываться следующие:

- функции прикладных ПС — системы управления объектами или процессами;
- прикладная область системы — оборудование и аппаратура управления процессами и объектами;
- режим эксплуатации — обработка данных в режиме реального времени;
- масштаб, объем ПС — средний или большой;
- представление данных — предметный или объектный;
- критичность ПС — высокая, возможность повреждение дорогой собственности или угроза человеческой жизни;
- класс пользователей — технические средства и квалифицированные специалисты;
- стабильность ПС — маловероятное или дискретное внесение изменений;

- готовность программного продукта — заказное для конкретного применения в системе;
- требуемые рабочие характеристики: емкость памяти — средняя или низкая; время отклика — быстрое, секунды; производительность — малая или средняя;
- требования безопасности и надежности — высокие;
- вычислительная система и среда — микропроцессорное управление и системы реального времени;
- требования к вычислительным ресурсам — высокие, почти полное использование ресурсов по основному функциональному назначению.

Требования функциональной безопасности относятся к достаточно узкому классу базовых и прикладных программных средств, которому соответствуют около 10 — 15% от общего числа типов характеристик, перечисленных в стандарте. Такие ПС применяются при управлении: движущимися объектами на транспорте и в авиации, в атомной энергетике, в автоматизированных производствах, во встроенных компьютерных системах и программируемых электронных системах безопасности (IEC 61508, IEC 60880). Характеристики и особенности выделенных типов ПС, для которых имеет важнейшее значение функциональная безопасность, могут быть несколько расширены свойствами комплексов программ информационных систем, в которых доминирующее значение имеет информационная безопасность, но отражающаяся на функциональной безопасности.

Однако далее учитываются базовые и прикладные ПС, преимущественно с перечисленными выше характеристиками. При этом предполагается, что функции обеспечения функциональной безопасности могут реализоваться автономными программами или органически входить в основные комплексы функциональных программ управления системой или процессами. Применение основных характеристик ПС в стандарте **ISO** 12182 иллюстрировано таблицей связей с характеристиками качества комплексов

программ, представленными в стандарте ISO 9126, и несколькими примерами.

Общее представление о характеристиках качества программных средств международным стандартом ISO 9126 рекомендуется отражать тремя взаимодействующими и взаимозависимыми метриками характеристик качества, описывающими:

- внутреннее качество, проявляющееся в процессах разработки, модификации и других промежуточных этапах жизненного цикла ПС;
- внешнее качество, заданное требованиями заказчика в спецификациях и отражающееся характеристиками конечного продукта;
- качество при использовании в процессе нормальной эксплуатации и результативностью достижения потребностей пользователей с учетом затрат ресурсов.

Эти типы метрик применимы при определении целей системы и требований к ПС, включая промежуточные компоненты и продукты. Подходящие внутренние атрибуты качества ПС являются предпосылкой достижения в жизненном цикле требуемого внешнего поведения, а приемлемое внешнее поведение — предпосылка достижения качества в использовании.

Стандартами рекомендуется, чтобы было предусмотрено измерение или оценивание каждой характеристики ПС (субхарактеристики или ее атрибута) с точностью и определенностью, достаточной для сравнений с требованиями технических заданий и спецификаций, и чтобы измерения были объективны и воспроизводимы. Следует предусматривать нормы допустимых ошибок измерения, вызванных инструментами и/или ошибками человека-эксперта. Чтобы измерения были объективными, должна быть документирована и согласована процедура для присвоения числового значения, свойства или категории каждому атрибуту программного продукта. Процедуры измерений должны давать в результате одинаковые меры с приемлемой устойчивостью,

получаемые различными субъектами при выполнении одних и тех же оценивании характеристик ПС.

Характеристики, субхарактеристики и атрибуты качества ПС с позиции возможности и точности их измерения можно разделить на три уровня детализации показателей, особенности которых следует уточнять при их выборе:

- категорийные-описательные, отражающие набор свойств и общие характеристики объекта — его функции, категории ответственности, безопасности и важности, которые могут быть представлены номинальной шкалой — категорий-свойств;
- количественные — представляемые множеством упорядоченных числовых точек, отражающих непрерывные или дискретные закономерности и описываемые интервальной или относительной шкалой, которые можно объективно измерить и численно сопоставить с требованиями;
- качественные — содержащие несколько упорядоченных или отдельных свойств — категорий, которые характеризуются порядковой или точечной шкалой набора категорий (есть — нет, хорошо — плохо), устанавливаются, выбираются и оцениваются в значительной степени субъективно и экспертно.

К **первому уровню** относятся показатели качества, которые характеризуются наибольшим разнообразием значений — свойств программ и наборов данных и охватывают весь спектр классов, назначений и функций современных ПС. Эти свойства можно сравнивать только в пределах однотипных ПС и трудно упорядочивать по принципу предпочтительности. Среди стандартизированных показателей качества к этой группе, прежде всего, относится Функциональная пригодность, являющаяся самой важной и доминирующей характеристикой любых ПС. Номенклатура и значения всех остальных показателей качества непосредственно определяются требуемыми функциями программного средства и, в той или иной степени, влияют на выполнение этих функций. Поэтому

выбор функциональной пригодности ПС, подробное и корректное описание ее свойств являются основными исходными данными для установления при проектировании требуемых значений функциональной безопасности и всех остальных стандартизированных показателей качества.

Ко **второму уровню** показателей качества относятся достаточно достоверно и объективно измеряемые численные характеристики ПС. Значения этих конструктивных характеристик обычно в наибольшей степени влияют на функциональную пригодность и метрики в использовании ПС. Поэтому выбор и обоснование их требуемых значений должно проводиться наиболее достоверно уже при проектировании ПС. Их субхарактеристики могут быть описаны упорядоченными шкалами объективно измеряемых значений, требуемые численные величины которых могут быть установлены и выбраны заказчиками или пользователями ПС. Такими характеристиками являются корректность, функциональная безопасность, надежность и эффективность комплексов программ. Эти величины могут выбираться и фиксироваться в техническом задании или спецификации требований и сопровождаться методикой объективных, численных измерений при квалификационных испытаниях для сопоставления с требованиями.

Третий уровень стандартизированных показателей качества ПС трудно полностью описать измеряемыми количественными значениями и их некоторые субхарактеристики и атрибуты имеют описательный, качественный вид. В зависимости от функционального назначения ПС по согласованию с заказчиком можно определять экспертно степень необходимости (приоритет) этих свойств и бальные значения уровня реализации их атрибутов в жизненном цикле конкретного ПС.

Различия между ожидаемыми и полученными результатами функционирования программ могут быть следствием ошибок не только в созданных программах и данных, но и системных ошибок в первичных требованиях

спецификаций, явившихся исходной базой при создании ПС. Тем самым проявляется объективная реальность, заключающаяся в невозможности абсолютной корректности исходных спецификаций требований сложных ПС для проектирования. На практике в процессе разработки ПС исходные требования уточняются и детализируются по согласованию между заказчиком и разработчиком. Базой таких уточнений являются неформализованные представления и знания специалистов, а также результаты промежуточных этапов жизненного цикла. Однако установить ошибочность исходных данных и спецификаций еще труднее, чем обнаружить ошибки в созданных программах, так как принципиально отсутствуют формализованные данные, которые можно использовать как эталонные, и их заменяют неформализованные представления заказчиков и разработчиков.

Дефекты функционирования программных средств, не имеющие злоумышленных источников или последствий физических разрушений аппаратных компонентов, проявляются внешне как случайные, имеют разную природу и последствия. Полное устранение негативных воздействий и дефектов, отражающихся на безопасности и качестве функционирования сложных ПС, принципиально невозможно. Проблема состоит в выявлении факторов, от которых они зависят, в создании методов и средств уменьшения их влияния на функциональную пригодность ПС, а также в эффективном распределении ограниченных ресурсов для обеспечения необходимого качества функционирования комплекса программ, равнопрочного при всех реальных негативных воздействиях. Комплексное, скоординированное применение этих методов и средств в процессе создания, развития и применения ПС позволяет исключать проявления ряда негативных факторов или значительно ослаблять их влияние. Тем самым уровень достигаемой функциональной безопасности и качества функционирования ПС может быть предсказуемым и управляемым, непосредственно зависящим от

ресурсов, выделяемых на его достижение, а главное, от системы качества и эффективности технологий, используемых на всех этапах жизненного цикла ПС.

Для выбора при проектировании значений характеристик качества программных средств необходимо, прежде всего, установить диапазоны рациональные мер и шкал для каждой субхарактеристики и ее атрибутов, которые целесообразно использовать в качестве первичных ограничений их значений для реальных ПС. Далее должны быть разработаны процессы определения и представления в спецификациях проекта, требований к свойствам и атрибутам каждой характеристики качества. Эти требования должны учитывать реальные ограничения ресурсов, доступных для их достижения в ЖЦ ПС.

Решение этих задач должно быть направлено на обеспечение высокой функциональной пригодности ПС путем сбалансированного улучшения безопасности и остальных характеристик качества в условиях ограниченных ресурсов на ЖЦ. Для этого в процессе системного анализа при подготовке технического задания и требований спецификаций, значения атрибутов и характеристик безопасности и качества должны выбираться с учетом их влияния на функциональную пригодность. Излишне высокие требования к отдельным атрибутам качества, требующие для реализации больших дополнительных трудовых и вычислительных ресурсов, целесообразно снижать, если они слабо влияют на основные, функциональные характеристики ПС. Ориентирами могут служить диапазоны изменения атрибутов конструктивных характеристик качества ПС, границы количественных или качественных шкал которых сверху и снизу могут быть выбраны на основе следующих принципов:

- предельные значения характеристик функциональной безопасности и качества должны быть ограничены сверху допустимыми или рациональными затратами ресурсов на их достижение при разработке и совершенствовании системы и ПС;

- наибольшие допустимые затраты ресурсов, например, труда и времени, для реализации функциональной безопасности и конструктивных характеристик должны обеспечивать функциональную пригодность жизненного цикла системы и ПС на достаточно высоком уровне;
- допустимые наихудшие значения безопасности и отдельных конструктивных характеристик качества могут соответствовать значениям, при которых начинает снижаться функциональная пригодность при применении системы и ПС;
- ограниченные значения отдельных конструктивных характеристик качества не должны негативно отражаться на возможных высоких значениях других приоритетных характеристик.

Функциональная пригодность — наиболее ответственная, неопределенная, объективно трудно формализуемая и оцениваемая в проектах характеристика комплексов программ, которая значительно определяет требования к обеспечению функциональной безопасности системы и ПС. Области применения, номенклатура и функции комплексов программ охватывают столь разнообразные сферы деятельности человека, что невозможно полностью выделить и унифицировать достаточно ограниченное число атрибутов для выбора и сравнения этой характеристики у различных по назначению комплексов программ. Перед системным проектированием комплекса программ заказчиком должен описываться исходный набор свойств ПС (ISO 12182), которые в дальнейшем используются в качестве эталонов при формализации функциональной пригодности.

Функциональная пригодность — это набор и описания атрибутов, определяющих назначение, основные, необходимые и достаточные функции ПС, заданные техническим заданием и спецификациями требований заказчика или потенциального пользователя. В процессе проектирования комплекса программ атрибуты функциональной пригодности должны конкретизироваться в спецификациях на ПС в целом и на

компоненты. Атрибутами этой характеристики качества могут быть функциональная полнота решения заданного комплекса задач, степень покрытия функциональных требований спецификациями и их стабильность при совершенствовании ПС, число реализуемых требований заказчиков и т.д. Кроме них функциональную пригодность отражают множество различных специализированных критериев, которые тесно связаны с конкретными решаемыми задачами и сферой применения комплекса программ. Их можно рассматривать как частные критерии или как факторы, влияющие на основной показатель качества ПС.

Эта характеристика может значительно модифицироваться в жизненном цикле ПС и соответственно изменять конкретное содержание функций, которые подлежат применению и оцениванию. На последовательных этапах ЖЦ функции промежуточных продуктов (спецификаций компонентов, модулей, текстов программ и т.п.) должны оцениваться на соответствие описаниям в отдельных, частных документах. Это позволяет поэтапно формализовать применяемые субхарактеристики и атрибуты функциональной пригодности. Такими атрибутами могут быть: функциональная адекватность программ документам и декларированным требованиям, утвержденным заказчиком; степень покрытия тестами исходных требований; полнота и законченность реализации этих требований; точность выполнения требований детальных спецификаций на функциональные компоненты ПС.

Среди всего многообразия функциональных характеристик программных средств можно выделить две группы, одна из которых отражает разнообразные специфические особенности, связанные непосредственно с назначением, функциями и сферой применения ПС, а вторая — доступна для частичной унификации состава и структуры, а также для оценивания стандартизированными методами. Эта вторая группа характеризует ряд базовых, инвариантных свойств качества, которые позволяют определять некоторые субхарактеристики функциональной пригодности ПС при

разных конкретных целях и сферах применения. С этой позиции функциональная пригодность определяется качеством взаимосвязи и согласованности последовательных формулировок содержания и реализации основных фрагментов в цепочке стандартизированных требований технического задания на ПС: целей — назначения — функций — исходной информации — результатов для пользователей, определяющих, что:

- описание целей программного средства корректно переработано в подробное описание его назначения и внешней среды применения;
- назначение ПС полностью и корректно детализировано в требованиях к функциям комплекса программ и его компонентов;
- реализация требований к функциям ПС обеспечена достоверным и адекватным составом и содержанием исходной информации и свойств объектов внешней среды;
- реализация функций ПС способна подготавливать всю требуемую и достаточно корректную информацию для пользователей и объектов внешней среды.

Цель жизненного цикла или системная эффективность ПС может оцениваться, в основном, экспертно и является исходной для прослеживания всех последующих, производных свойств и атрибутов функциональной пригодности. Назначение ПС детализируются и формализуются в требованиях к функциям компонентов и всего комплекса программ, способного реализовать декларированные цели. Адекватность и полнота отражения требуемыми функциями сформулированного назначения ПС являются характеристикой, определяющей потенциальную возможность реализации его функциональной пригодности в целом. Прослеживание детализации и покрытия целей требованиями к функциям сверху вниз (начиная от целей системы), а также конкретизация и корректировка целей снизу-вверх от потенциально реализуемых функций должны обеспечивать

адекватность и качество этой части декларируемой основы функциональной пригодности.

Функции программного средства реализуются в определенной аппаратной, операционной и пользовательской внешней среде системы, характеристики которой существенно влияют на функциональную пригодность. Для выполнения требуемых функций комплекса программ необходима адекватная исходная информация от объектов внешней среды, содержание которой должно полностью обеспечивать реализацию декларированных функций. Полнота формализации номенклатуры, структуры и качества входной информации для выполнения требуемых функций, является одной из важных составляющих при определении функциональной пригодности ПС в соответствующей внешней среде.

Цель и функции ПС реализуются тогда, когда выходная информация достигает потребителей — объектов или операторов-пользователей с требуемым содержанием и качеством, достаточным для обеспечения её эффективного применения. Содержательная часть этой информации определяется конкретными задачами системы, их основными технико-экономическими и/или социальными показателями функционирования и отражается метриками в использовании. Степень покрытия всей выходной информацией: целей, назначения и функций ПС для пользователей, следует рассматривать как основную меру качества функциональной пригодности. Прослеживание и оценивание адекватности и полноты состава выходной информации снизу-вверх к назначению ПС должны завершать выбор базовых субхарактеристик качества функциональной пригодности, независимо от сферы применения системы.

В процессе проектирования в составе функциональной пригодности могут быть выделены две группы базовых субхарактеристик, определяющие функциональные и структурные требования и особенности ПС. При формализации

и выборе функциональных требований следует возможно четко формулировать в документах контракта:

- экономические, организационные, технические и/или социальные стратегические цели всего жизненного цикла системы, ПС и его компонентов;
- системную эффективность и, в том числе, требуемые технико-экономические показатели применения ПС в составе системы;
- назначение, внешнюю среду, условия эффективного и безопасного применения ПС;
- функциональные задачи основных компонентов и ПС в целом, а также системную эффективность каждого компонента;
- необходимую и достаточную безопасность применения, характеристики качества и временной регламент решения каждой функциональной задачи;
- соответствие ПС и его компонентов выделенным стандартам и нормативным документам на проектирование и применение.

В зависимости от назначения ПС функциональная безопасность и/или некоторая конструктивная характеристика может стать доминирующей или даже почти полностью определяющей функциональную пригодность ПС. В наибольшей степени функциональная пригодность во многих случаях зависит от функциональной безопасности, корректности и надежности ПС. Эти характеристики трудно свести к количественным мерам и зачастую их приходится оценивать по наличию свойств и ряда типовых процедур в ПС или по величине необходимых затрат ресурсов, достаточно заметно влияющих на функциональную пригодность.

Правильность — корректность: это способность ПС обеспечивать правильные или приемлемые по качеству результаты для пользователей. Эталонами для выбора требований к корректности при проектировании могут быть: верифицированные и взаимоувязанные требования к функциям комплекса, компонентов и модулей программ, а также правила

их структурного построения, организация взаимодействия и интерфейсов. Эти требования к ПС при разработке должны быть прослежены сверху вниз до модулей и использоваться как эталоны при установлении необходимой корректности соответствующих компонентов. Данное понятие включает обеспечение эталонных (ожидаемых) данных с необходимой степенью точности расчетных значений в соответствии с требованиями технического задания и спецификаций. В процессе проектирования и разработки модулей и групп программ применяются частные структурные критерии корректности, которые включают корректность структуры программ, обработки данных и межмодульных интерфейсов. Каждый из частных критериев может характеризоваться несколькими методами измерения качества и достигаемой степенью корректности программ: детерминировано, стохастически или в реальном времени.

Требования к характеристике корректность могут представляться в виде описания двух основных свойств, которым должны соответствовать все программные компоненты и ПС в целом. Первое требование состоит в выполнении определенной степени (%) прослеживаемости и верификации сверху вниз реализации требований технического задания и спецификации на ПС при последовательной детализации описаний программных компонентов вплоть до текстов и объектного кода программ.

Второе требование заключается в выборе степени и стратегии покрытия тестами структуры и функций программных компонентов, совокупности маршрутов исполнения модулей и всего комплекса программ для последующего процесса верификации и тестирования, достаточного для функционирования ПС с необходимым качеством и точностью результатов при реальных ограничениях ресурсов. Для определения этой величины при разработке ПС необходима организация регулярной регистрации, накопления имен, содержания функций и маршрутов исполнения программ, прошедших тестирование, а

также контроль доли не тестированных от всей совокупности. Мерой выбранной корректности может быть относительное число протестированных функций и маршрутов, которое может измеряться в процентах от общего числа исполняемых. Опыт показывает, что зачастую в готовом, сложном ПС оказываются протестированными только около 50-70% функций и маршрутов, и практически очень трудно эту величину довести до 90-95%. Косвенно эту величину при определенной автоматизации и квалификации специалистов отражает трудоемкость и длительность тестирования, что непосредственно влияет на функциональную пригодность ПС.

3.4. Ресурсы для обеспечения функциональной безопасности программных средств

Многие проекты обеспечения безопасности систем терпели и терпят неудачу из-за отсутствия у разработчиков и заказчиков при подготовке контракта четкого представления о реальных финансовых, трудовых, временных и иных ресурсах, необходимых и доступных для их реализации. Поэтому одной из основных задач при проектировании функциональной безопасности ПС является экономический анализ и определение необходимых ресурсов для создания и всего ЖЦ ПС в соответствии с требованиями контракта и технического задания.

При планировании проектов ПС часто инициатором разработки является разработчик-поставщик, который самостоятельно принимает все решения о проектировании за счет собственных ресурсов и предполагает возместить затраты при реализации ПС на рынке. В других случаях имеется определенный заказчик-потребитель, который способен задать основные цели, характеристики качества и безопасности и обеспечить ресурсы для реализации проекта. Таким образом, при экономическом обосновании проектов ПС и их функциональной безопасности возможны два сценария:

- создание и весь жизненный цикл комплекса программ и/или базы данных ориентируется разработчиком на массовое тиражирование и распространение на рынке для заранее неизвестных покупателей-пользователей в различных сферах применения, при этом отсутствует приоритетный внешний потребитель-заказчик, который определяет и диктует основные требования к ПС и его безопасности, а также финансирует проект;
- разработка проекта ПС и/или БД и их безопасности предполагается поставщиком-разработчиком для конкретного потребителя-заказчика, задающего все требования, который его финансирует, с определенным, необходимым ему тиражом и известной ограниченной областью применения результатов разработки.

Первый сценарий базируется на маркетинговых исследованиях рынка программных продуктов и на стремлении поставщика занять на рынке достаточно выгодное место, обеспечивающее ему необходимую прибыль. Важнейшим фактором конкурентоспособности ПС является соотношение между ценностью (эффективностью) имеющегося или предполагаемого продукта с позиции его использования потребителем и стоимостью его при создании или приобретении в условиях реального рынка. Для этого нужно определить наличие на рынке гаммы близких по назначению ПС, оценить их экономическую эффективность, стоимость, применяемость и безопасность, а также возможную конкурентоспособность предполагаемого программного продукта для потенциальных пользователей и их возможное число. Кроме того, следует оценить рентабельность затрат на обеспечение всего ЖЦ нового ПС и выявить функциональные и конструктивные характеристики качества и безопасности, которые способны привлечь достаточно покупателей и оправдать затраты на предстоящую разработку.

Второй сценарий предполагает наличие определенного заказчика-потребителя проекта ПС, который определяет

основные технические и экономические требования и функциональные характеристики. Он выбирает конкурентоспособного поставщика-разработчика, которого оценивает на возможность реализовать проект с необходимым качеством и функциональной безопасностью с учетом ограничения сроков, бюджета и других ресурсов. Этому помогает опыт и экономические характеристики ранее выполненных поставщиками проектов, но некоторые проекты могут не иметь явных прецедентов, и тогда приходится использовать имеющуюся статистику в этой области. При этом предполагается, что результаты разработки могут не поступать на открытый рынок, вследствие чего маркетинговые исследования для таких проектов не являются доминирующими и обычно предварительно могут не проводиться.

При прогнозировании необходимых ресурсов для проекта, ЖЦ ПС и обеспечение его безопасности можно разделить на две части, существенно различающиеся экономическими особенностями процессов, характеристиками и влияющими на них факторами. В первой части ЖЦ производятся системный анализ, проектирование, разработка, тестирование и испытания базовой версии ПС. Номенклатура работ, их трудоемкость, длительность и другие экономические характеристики на этих этапах ЖЦ существенно зависят от требуемых характеристик системы и ее безопасности, технологии и инструментальной среды разработки.

Вторая часть ЖЦ ПС, отражающая эксплуатацию, сопровождение, модификацию и перенос ПС на иные платформы, в меньшей степени связана с функциональными характеристиками системы и среды разработки. Номенклатура работ на этих этапах более или менее определенная и стандартизированная (ISO 12207, ISO 14764, ISO 15846), но их трудоемкость и длительность могут сильно варьироваться в зависимости от массовости и других внешних факторов распространения и применения версий ПС. Определяющими становятся потребительские характеристики ПС, а их

экономические особенности с позиции разработчиков и вложенные затраты на очередную версию отходят на второй план (см. первый сценарий). Поэтому планы на этих этапах имеют характер общих взаимосвязей содержания работ, которые требуют распределения во времени индивидуально для каждого проекта.

Важнейшим ресурсом при создании любых ПС являются люди — специалисты с их уровнем профессиональной квалификации, а также с многообразием знаний, опыта, стимулов и потребностей. Быстрый рост сложности и повышение ответственности за качество и функциональную безопасность комплексов программ привели к появлению новых требований к специалистам, обеспечивающим все этапы жизненного цикла ПС. Эти специалисты должны владеть новой интеллектуальной профессией, обеспечивающей высокое качество и безопасность ЖЦ ПС, а также контроль, испытания и удостоверение реального достигнутого качества на каждом этапе разработки и совершенствования программ.

Безопасность вычислительной системы с ПС обеспечивается двумя видами работ: созданием функциональных программ высокого качества с минимальным количеством дефектов и ошибок, отражающихся на безопасности, и разработкой специального комплекса программ, повышающих безопасность функционирования основных программ. Руководство безопасностью сложного проекта ПС должны осуществлять лидеры — менеджеры:

- менеджер безопасности проекта — это специалист, обеспечивающий коммуникацию между заказчиком и проектной командой специалистов, его задача — определить и обеспечить полное удовлетворение требований заказчика по безопасности системы и ПС;
- менеджер-архитектор комплекса программ повышения безопасности — управляет коммуникациями и взаимоотношениями в проектной команде, является координатором создания компонентов, разрабатывает

базовые, функциональные спецификации и управляет ими, ведет график проекта и отчитывается за его состояние, инициирует принятие критичных для хода проекта решений.

Функции специалистов и технология работы при обеспечении ЖЦ комплекса программ повышения безопасности ПС аналогичны тем, которые применяются при создании основных компонентов, определяющих функциональную пригодность ПС и системы. В реализации любого крупного проекта ПС можно выделить две категории специалистов: разрабатывающих компоненты и ПС в целом и обеспечивающие технологию, безопасность и качество программного продукта. При выборе заказчиком надежного поставщика-разработчика проекта необходима оценка тематической и технологической квалификации возможного коллектива специалистов, а также его способности реализовать проект с заданными требованиями, качеством и функциональной безопасностью.

Специалисты первой категории непосредственно создают компоненты и ПС в целом с заданными показателями качества и безопасности. В процессе разработки их функции заключаются в тщательном соблюдении принятой в фирме технологии и в формировании всех предписанных руководствами исходных и отчетных документов. Разделение труда специалистов этой категории в крупных проектных коллективах приводит к необходимости их дифференциации по квалификации и областям деятельности:

- спецификаторы — подготавливают описания функций соответствующих компонентов с уровнем детализации, достаточным для корректной разработки текстов программ программистами;
- разработчики программных компонентов (программисты) создают компоненты, удовлетворяющие спецификациям, реализуют требуемые функции продукта;
- системные интеграторы создают на выходе требуемые крупные компоненты или комплекс программ;

- тестировщики — обеспечивают проверку функциональных спецификаций, выполняют тестирование для каждой из фаз и компонента проекта;
- управляющие сопровождением и конфигурацией — обеспечивают взаимодействие компонентов и реализацию версий ПС;
- документаторы — осуществляют подготовку и издание сводных технологических и эксплуатационных документов в соответствие с требованиями стандартов.

Специалисты второй категории — технологи, обслуживающие и сопровождающие технологический инструментарий, который применяется специалистами первой категории, обеспечивают применение системы качества проекта или предприятия, контролируют и инспектируют ее использование. Специалисты, управляющие обеспечением качества и безопасности ПС, должны овладеть стандартами и методиками фирмы, поддерживающими регистрацию, контроль, документирование и воздействия на показатели качества на всех этапах ЖЦ программ. Они должны обеспечивать эксплуатацию системы качества проекта, выявление всех отклонений от заданных показателей качества и безопасности объектов и процессов, а также от предписанной технологии на промежуточных и заключительных этапах разработки.

Для анализа затрат ресурсов в жизненном цикле ПС их целесообразно разделить на две части (рис. 3.4):

- затраты на создание программных компонентов, обеспечивающих базовые свойства функциональной пригодности комплекса программ для его применения по прямому назначению пользователями в соответствии с требованиями контракта и технического задания;
- основные составляющие дополнительных затрат, обеспечивающие требуемые конструктивные характеристики качества и безопасности для улучшения функциональной пригодности ПС в соответствии с целями и сферой его применения.

Рис. 3.4. Классификация затрат ресурсов в жизненном цикле ПС

Обеспечение функциональной пригодности является основной целью при использовании финансовых, трудовых, вычислительных и других ресурсов в жизненном цикле ПС. Эти затраты зависят, в первом приближении, от сложности алгоритмов, объема комплекса программ и баз данных, которые определяют трудоемкость и длительность полного цикла их разработки. Основные затраты идут на овеществленный, преимущественно интеллектуальный труд специалистов различных категорий. Поэтому для их измерения наиболее универсальной единицей стали трудозатраты специалистов в человеко-днях или человеко-месяцах, которые обычно достаточно просто могут преобразовываться в стоимость процесса разработки. Однако это не значит, что затраты на решение этой основной задачи всегда являются доминирующими по величине. Необходимость выполнения ряда требований к остальным конструктивным характеристикам качества и безопасности часто приводит к тому, что использование ресурсов на их реализацию может превышать базовые затраты на обеспечение функциональной пригодности. В то же время затраты на выполнение этих требований всегда направлены на повышение и совершенствование функциональной пригодности.

Абсолютная величина затрат на разработку ПС, также, как и ее длительность, зависят от ряда факторов, которые могут изменять их в различных направлениях. При анализе затрат на обеспечение требуемых функций сложных ПС доминирующим фактором, влияющим на их величину, является сложность функциональной части комплекса программ и базы данных. Наиболее активно в качестве простейшего показателя сложности используется объем программ, выраженный числом операторов (команд) или строк текста на языке программирования (с учетом коэффициента, зависящего от класса ПС и специфики языка). Объем программ, без комментариев, является одной из наиболее достоверно измеряемых характеристик сложности ПС и достаточно адекватен экономическим затратам на его разработку. Реальное

изменение создаваемых в настоящее время новых сложных ПС объемом от 10 3 до 10 6 строк определяет диапазон трудоемкости разработки таких программ от человеко-года до тысяч человеко-лет.

Ограниченные ресурсы времени реализации проектов ПС являются одним из самых сильных факторов, влияющих на достижимое качество и безопасность комплексов программ. При реальном допустимом времени реализации проекта оно ограничивает затраты на все промежуточные этапы работ и тем самым на качество их выполнения. При современных технологиях полностью новые, сложные комплексы программ, реализуемые в допустимое время 2—4 года, ограничены объемом 10 6 —10 7 строк текста. Очевидна принципиальная нерентабельность разработки очень сложных ПС за 5—10 лет. С другой стороны, даже относительно небольшие программы высокого качества в несколько тысяч строк по полному технологическому циклу с сертификационными испытаниями продукции редко создаются за время, меньшее, чем полгода — год. Таким образом, диапазон вариации длительностей разработок ПС много меньше, чем вариация их трудоемкости, а эти длительности ограничены сверху и снизу, и объем новых программ является одним из основных факторов, определяющим эти границы.

Любые ПС должны поступать на эксплуатацию, сохраняя актуальность до того, как в них пропадает необходимость. Их цели, концептуальная основа и алгоритмы не должны устареть за время разработки. Отсюда появляется верхний предел допустимых длительностей разработки. Стремление заказчиков ограничивать длительность реальных разработок ПС приводит к объективному формированию ее верхнего предела (2 — 4 года), зависящего в основном от объема и класса ПС, за которым распространяется зона «нерациональных» длительностей.

Границу длительностей снизу определяют естественный технологический процесс коллективной разработки и необходимость выполнения ряда взаимоувязанных работ на

последовательных этапах, которые обеспечивают получение сложных ПС требуемого качества. Подготовка текстов программ, их тестирование, комплексирование, документирование и испытания могут проводиться в основном последовательно, и каждый этап требует некоторого времени. Под воздействием перечисленных факторов формируется объективный минимум возможных длительностей разработок — граница снизу области «невозможных» длительностей, зависящая в первую очередь от объема разрабатываемых ПС. Эта граница может варьироваться в небольших пределах (1 — 2 года для сложных ПС) за счет: совершенствования технологии, повышения программной и аппаратурной оснащенности разработки, а также вследствие роста коллективной квалификации разработчиков и заказчиков.

Основные составляющие дополнительных затрат, обеспечивающих требуемые характеристики функциональной безопасности и качества ПС, целесообразно структурировать в соответствии с их номенклатурой в стандарте ISO 9126. Эти затраты обычно состоят из двух связанных частей: из затрат на реализацию соответствующих характеристик качества в программных продуктах и из затрат при использовании этих характеристик в процессе эксплуатации комплекса программ. Важнейшее значение имеет установление и формализация исходных требований к характеристикам качества и безопасности ПС. Поэтому целесообразно выделять ресурсы на системный анализ и проектирование требований ко всему комплексу характеристик безопасности и качества на начальных этапах проекта ПС. Обычно совершенствование качества и повышение затрат на реализацию характеристик способствует снижению затрат при их эксплуатации.

Затраты ресурсов на обеспечение корректности зависят от полноты прослеживания реализации требований к ПС сверху вниз, от требований к компонентам вплоть до объектного кода программ и от степени их покрытия тестами. Эти затраты входят непосредственно в процесс разработки и зависят от объема и детальности процессов верификации и

тестирования. Для сложных ПС при требовании их высокой корректности они могут составлять до 30% от затрат на обеспечение базовой, функциональной пригодности. Для относительно простых комплексов программ эта величина в среднем составляет 10—20%. Эти затраты приходятся на верификацию и тестирование программных компонентов, что должно обеспечивать корректность, безопасность ПС и надежность в целом. Хотя эти характеристики различаются, затраты на их реализацию полностью разделить невозможно. Поэтому оценивание и выделение ресурсов на решение этих задач целесообразно анализировать совместно.

Затраты на квалификационное тестирование и испытания ПС в целом обычно могут быть достаточно четко выделены из остальных затрат, так как в этих процессах непосредственно участвуют заказчик и пользователи. Величина этих затрат без учета ресурсов, необходимых для имитации внешней среды, может составлять около 10% от общих затрат на разработку. При этом практически невозможно разделять затраты на оценивание отдельных стандартизированных характеристик.

Затраты на функциональную безопасность и надежность ПС определяются требуемым уровнем защищенности и сложностью (объемом) программ для ее реализации. При наличии особенно высоких требований к безопасности критических ПС эти затраты могут даже в 2 — 3 раза превышать затраты на решение базовых, функциональных задач. Для типовых систем трудоемкость создания программных средств функциональной безопасности обычно составляет 20 — 40% затрат на решение основных, функциональных задач при требованиях наработки на отказ в десятки тысяч часов и для минимального обеспечения автоматического рестарта в системах.

Возможные затраты трудовых и временных ресурсов на развитие и совершенствование качества и безопасности комплекса программ в процессе сопровождения зависят не только от внутренних свойств программ, но также от запросов

и потребностей пользователей на новые функции и от готовности заказчика и разработчика удовлетворить эти потребности. Величина этих затрат определяется рыночной конъюнктурой для данного программного продукта и коммерческой целесообразностью его модификации и развития. По объему предполагаемых изменений, а также вновь вводимых в очередную версию компонентов, с учетом сложности и новизны их разработки могут быть оценены затраты на их создание.

Затраты на обеспечение и реализацию сопровождения программ определяются длительностью цикла жизни комплекса программ, его мобильностью, уровнем автоматизации технологии разработки и тиражом программ. Для их оценивания, прежде всего, необходимо выделять основные виды затрат при сопровождении конкретного комплекса программ и наиболее существенные факторы, которые на них влияют. Сокращение затрат на сопровождение возможно за счет некоторого увеличения затрат при разработке ПС, так что при рациональном проектировании в сумме затраты могут быть уменьшены иногда весьма заметно. Затраты на сопровождение можно считать аддитивными и включающими составляющие:

- затраты на обнаружение и устранение ошибок и дефектов в каждой версии ПС;
- затраты на доработку и совершенствование программ, формирование и испытание новых модернизированных версий ПС;
- затраты на тиражирование каждой новой версии и ее внедрение в эксплуатируемых и новых ПС.

В современных проектах ПС большую или меньшую долю составляют готовые апробированные компоненты из других подобных разработок — прототипы и/или покупные пакеты прикладных программ. Это позволяет значительно ускорять работы и сокращать затраты на создание сложных комплексов программ. Перед разработчиками проекта ПС зачастую возникает дилемма: разрабатывать ли весь комплекс

программ (в том числе для обеспечения безопасности) полностью из новых компонентов или использовать, адаптировать и приспосабливать готовые компоненты (какие и в каком количестве). В результате при первичном экономическом анализе затрат на создание ПС с требуемыми характеристиками безопасности целесообразно рассматривать два альтернативных варианта определения затрат на разработку:

- полностью нового ПС, для которого отсутствуют или недоступны подходящие готовые компоненты — прототипы и/или их заведомо нерентабельно использовать;
- программного средства на базе комплексирования набора готовых программных компонентов и информации баз данных, для которого почти не требуется создания новых компонентов.

При сборке нового ПС из комплектующих компонентов может потребоваться доработка некоторых из них или создание специальных программ для их взаимодействия. В пределе при построении нового ПС на 80—90% из готовых компонентов суммарные затраты могут сокращаться в 3—5 раз. В этом случае кроме 10—20% затрат на создание новых программных компонентов, необходимы ресурсы на комплексирование нового ПС, его квалификационное тестирование, испытания и документирование.

При создании сложных ПС одной из больших составляющих затрат могут быть ресурсы на генерацию тестов. В ряде случаев они соизмеримы с затратами на создание основных функций комплексов программ (в том числе для обеспечения безопасности), что определяется принципиальным соответствием сложности необходимых наборов тестов и тестового покрытия программ, и сложности функций, реализуемых испытываемым ПС. Создание представительных совокупностей тестов возможно путем использования реальных объектов внешней среды или с помощью программных имитаторов, адекватных этим объектам по

результатам функционирования и генерируемой информации. При этом возникает проблема — какой метод и когда выгодней по затратам на генерацию тестов и по обеспечению необходимой степени покрытия тестами испытываемых ПС. Затраты ресурсов на натурные эксперименты для генерации тестов при проведении разработки, испытаний и определения качества пропорциональны реальному времени функционирования проверяемого ПС и затратам на применение привлекаемых средств реальной внешней среды. Они включают стоимость эксплуатации реального объекта, создающего тесты в единицу времени.

Имитаторы тестов необходимы не только для оценивания достигнутых характеристик безопасности и качества комплексов программ, но также для их комплексной отладки, квалификационного тестирования, испытаний при создании версий. Поэтому затраты на программные имитаторы и их экономическую эффективность целесообразно рассматривать с учетом всего комплекса задач, которые они способны и должны решать в ЖЦ ПС. Затраты на программную имитацию тестовых данных определяются ресурсами, необходимыми на проектирование и эксплуатацию сложных комплексов программ для этих целей и следующими составляющими:

- затратами на разработку комплекса программ для имитации информации внешних объектов и среды их функционирования;
- затратами на эксплуатацию программ имитации за время проведения тестирования, испытаний и/или определения характеристик безопасности и качества тестируемого ПС;
- затратами на первичную установку и эксплуатацию моделирующей ЭВМ и вспомогательного оборудования, используемого в имитационном стенде.

Затраты на эксплуатацию программ имитации в основном определяются длительностью проведения тестирования, испытаний и/или измерения характеристик

безопасности и качества ПС. Значения этого времени соответствуют реальному времени генерации тестовых данных и тестирования программ. Обычно имитаторы используются для тестирования нескольких ПС разного, но близкого целевого назначения. В результате удельные затраты на их создание и эксплуатацию быстро убывают при унификации имитаторов и расширении области их применения для тестирования и оценивания качества большого числа ПС, имеющих близкое функциональное назначение. Даже приближенные оценки этих затрат в большинстве случаев показывают высокую рентабельность программных имитаторов внешней среды, особенно для квалификационного тестирования и оценивания характеристик безопасности сложных ПС реального времени.

4. ОБЕСПЕЧЕНИЕ БЕЗОПАСНОЙ РАЗРАБОТКИ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

4.1. Нормативно-правовые основы обеспечения безопасной разработки программного обеспечения

На протяжении последних лет наблюдается устойчивый рост количества нарушений ИБ, приводящих к снижению уровня целостности, доступности и конфиденциальности информационных ресурсов автоматизированных систем. Опыт работы аккредитованной испытательной лаборатории, проводящей сертификационные испытаний ПО в различных системах сертификации, показывает, что даже в продуктах, представляемых разработчиками на сертификацию, содержатся уязвимости и дефекты безопасности. Наиболее распространенными типами уязвимостей, выявляемых в ПО, подаваемом сертификацию, являются: аутентификационные данные в исходном коде ПО, межсайтовый скрптинг, внедрение SQL-кода.

Проведенные исследования позволили определить три основных причины наличия уязвимостей в ПО.

1. Недостаток мотивации разработчиков. Потребители в первую очередь приобретают ПО, которое является первым на рынке или обладает уникальными функциональными возможностями. В связи с этим разработчики ПО заинтересованы в скорейшем выпуске ПО или его обновлений на рынок, что влечет за собой отсутствие должного внимания к вопросам обеспечения безопасности.

2. Отсутствие у разработчиков необходимых знаний. Методы построении безопасного ПО еще находиться в стадии изучения и формирования. В связи с этим современный разработчик ПО не всегда полностью понимает, как правильно проектировать и разрабатывать программу именно с точки зрения безопасности, что ведет к наличию большого числа уязвимостей в ПО, выпускаемом на рынок.

3. Отсутствие у разработчиков необходимых технологий. Современные инструментальные средства обеспечения безопасности разработки (статические анализаторы, сканеры уязвимостей и т.д.) еще не вышли на необходимый уровень развития и не еще могут в полной мере помочь разработчикам решить вопросы безопасности их ПО.

Угрозы безопасности ПО могут быть классифицированы по стадии жизненного цикла разработки ПО (стадии разработки, поставки, установки и эксплуатации), для которой они характерны.

Источником угроз безопасности ПО на стадии разработки ПО являются внутренние нарушители, в первую очередь недобросовестные разработчики, которые пытаются повлиять на штатный режим функционирования ПО путем внесения несанкционированных изменений в следующие объекты:

- требования к ПО, проектная документация ПО;
- исходные тексты ПО или исполняемые модули;
- тестовая документация или результаты тестирования ПО;
- эксплуатационная документация;
- инструментальные средства, используемые при разработке ПО.

Недобросовестные разработчики могут как входить в штат компании-разработчика ПО или привлекаться временно, так и являться разработчиком ПО с открытым исходным текстом, которое используется при реализации ПО.

Источником угроз безопасности ПО на стадии поставки ПО могут являться организации, осуществляющие поставку ПО конечному пользователю. В ходе этой процедуры в ПО могут быть несанкционированно внедрены вредоносные модули или выполнена замена стандартных модулей ПО на модули с не декларированными возможностями. Если поставка ПО осуществляется с использованием сетевых протоколов передачи данных (например, путем получения дистрибутива

ПО с сервера организации-разработчика), то источником угрозы может выступать внешний нарушитель, который может заменить файлы дистрибутива ПО в процессе их передачи или перенаправить пользователя на вредоносный сайт для скачивания дистрибутива ПО с использованием атак типа «межсайтовый скриптинг».

Источником угроз безопасности ПО на стадии эксплуатации ПО являются внутренние или внешние нарушители, которые могут выполнять локальные или удаленные (с использованием сетевых протоколов) атаки на ПО с целью эксплуатации уязвимостей.

Таким образом, на сегодняшний день важным направлением развития, нацеленным на повышение качества ПО и уменьшение числа уязвимостей и дефектов безопасности, является внедрение цикла разработки безопасного ПО. Опыт компании Microsoft показал, что внедрение цикла разработки безопасного ПО позволяет сократить число уязвимостей в ПО компании в среднем на 80%. В настоящее время известны методологии проектирования безопасного ПО, которые успешно используются предприятиями-разработчиками.

В настоящее время ФСТЭК России ведется работа над государственным стандартом «Защита информации. Требования по обеспечению безопасности разработки программного обеспечения». Кроме этого, в соответствии с нормативным правовым актом ФСТЭК России «Состав и содержание организационных и технических мер по обеспечению безопасности персональных данных при их обработке в информационных системах персональных данных», использование в информационной системе системного и (или) прикладного ПО, разработанного с использованием методов защищенного программирования, является дополнительной мерой по обеспечению безопасности информации в случае определения в качестве актуальных угроз безопасности персональных данных 1-го и 2-го типов. В тоже время документ, определяющий содержание и порядок выполнения работ по созданию ПО с использованием методов

защищенного программирования, отсутствует. Эти положения и определяют актуальность разработки какого-либо проекта в государстве.

При разработке первой редакции проекта учитывались следующие особенности:

1. Поскольку известно, что стоимость устранения уязвимостей и дефектов безопасности ПО выше на поздних стадиях проектирования, проект должен обеспечить внедрение необходимых процедур на самых ранних стадиях проектирования ПО.

2. Проект должен учитывать современные тенденции разработки безопасных ПО, учитывать положения «лучших практик» (например, Microsoft SDL, Cisco SDL).

3. Проект должен быть полностью совместим с методологией разработки и сертификации ПО, используемой в настоящее время ФСТЭК России («Общие критерии»).

4. Разрабатываемый проект должен обеспечить возможность интеграции процедур разработки безопасного ПО с существующей на предприятии системой управления ИБ.

Анализ современных методологий проектирования безопасного ПО позволил сформулировать следующий перечень процедур, которые должны быть реализованы разработчиком для успешного противостояния угрозам безопасности ПО:

- процедуры управления конфигурацией
- процедуры определения модели жизненного цикла;
- процедуры проектирования и реализации безопасных ПО;
- процедуры использования инструментальных средств и методов разработки;
- процедуры обеспечения безопасности разработки;
- процедуры поставки;
- процедуры обновления и устранения уязвимостей и дефектов безопасности ПО.

Отдельно рассмотрим перечень основных требований к процедурам проектирования и реализации безопасных ПО, сформулированные по результатам анализа «лучших практик» и обобщения опыта работы аккредитованных испытательных лабораторий систем сертификации средств защиты информации.

1. Должно проводиться периодическое обучение разработчиков ПО с целью повышения их осведомленности в области разработки безопасного ПО. Выполнение данного требования позволяет обеспечить повышение осведомленности в области безопасной разработки ПО сотрудников, связанных с разработкой ПО и, как следствие, повышение уровня безопасности разрабатываемого ПО. В программу обучения сотрудников могут входит курсы безопасного программирования, инспекции кода, тестирования на проникновение, статического анализа, динамического анализа, функционального тестирования и другие.

2. Разработчиком ПО должны быть определены и документированы требования безопасности к разрабатываемому ПО. Например, могут быть определены следующие классы требований: требования к обеспечению конфиденциальности, требования к обеспечению идентификации и аутентификации,

требования к реализации разграничения доступа, требования к обработке ошибок и исключений ПО.

3. Проектирование ПО должно выполняется с использованием принципов проектирования безопасных ПО. При выполнении проектирования разработчиком могут, например, использоваться следующие принципы: принцип эшелонированной защиты, принцип минимальных привилегий, принцип модульного проектирования, принцип разделения обязанностей.

4. При проектировании ПО разработчиком должно выполняться моделирование угроз с целью выявления уязвимостей ПО этапа проектирования, результаты моделирования угроз должны документироваться.

5. Разработка ПО должна выполняться с использованием методов защищенного программирования. Например, в ходе разработки ПО разработчик должен избегать использования скомпрометированных (небезопасных) функций в исходных кодах ПО. В качестве источника небезопасных функций могут использоваться, например, списки «Security Development Lifecycle Banned Function Calls» компании Microsoft.

6. Должен проводиться периодический статический анализ исходных кодов ПО с целью выявления уязвимостей и дефектов кода, результаты анализа должны документироваться.

7. Должна проводиться периодическая инспекция кода с целью выявления уязвимостей и дефектов кода, результаты инспекции должны документироваться.

8. Должно проводиться и документироваться тестирование (функциональное, нагрузочное) ПО.

9. Должен проводиться динамический анализ исходных кодов ПО с целью выявления уязвимостей и дефектов кода, результаты анализа должны документироваться.

10. Должно проводиться тестирование на проникновение с целью выявления уязвимостей ПО, результаты тестирования должны документироваться.

11. Должны быть разработаны функциональная спецификация ПО, проектная документация (проект верхнего уровня, проект нижнего уровня) и документация, демонстрирующая соответствие между всеми смежными парами имеющихся представлений ПО. Планируется, что будет введена некоторая градация, которая позволит предъявлять требования к организациям-разработчикам в зависимости от критичности создаваемого ПО.

Внедрение подобных процедур в российские организации-разработчики ПО, на наш взгляд, повысить уровень защищенности создаваемого ПО и, как следствие, значительно уменьшить число инцидентов ИБ.

4.2. Термины и определения

В настоящем стандарте применены следующие термины с соответствующими определениями:

1. **Дефект безопасности программы**: Недостаток программы, появившийся в результате ее разработки без учета требований по безопасности информации или в случае наличия ошибок проектирования и/или реализации.

2. **Динамический анализ программы**: Вид работ по поиску дефектов безопасности программы, основанный на анализе программы в режиме непосредственного исполнения (функционирования) с доступом или без доступа к исходным модулям.

3. З**ащищенное программирование**: Совокупность процедур, используемая организацией-разработчиком ПО для устранения (уменьшения количества) дефектов безопасности программы.

4. Н**агрузочное тестирование**: Вид работ по выявлению дефектов безопасности программы, основанный на наблюдении за потенциально опасными состояниями программы, возникающими в процессе увеличения количества внешних запросов на обслуживание.

5. **Инспекция исходных модулей**: Вид работ по выявлению дефектов безопасности программы, основанный на экспертизе исходных модулей программы.

6. **Инструментальные средства разработки**: Инструментальные средства (включая программные средства тестирования, если применимы), поддерживающие разработку и производство ПО.

7. И**сходный модуль:** Программный модуль на исходном языке, обрабатываемый транслятором и представляемый для него как целое, достаточное для проведения трансляции.

8**. Организация-разработчик (программного обеспечения):** Организация, которая выполняет разработку ПО в процессе жизненного цикла.

9**. План управления конфигурациями:** Описание порядка использования системы управления конфигурациями для конкретного ПО.

10. **Потребитель (программного обеспечения)**: Юридическое или физическое лицо, применяющее ПО.

11. **Программа**: Данные, предназначенные для управления конкретными компонентами системы обработки информации в целях реализации определенного алгоритма.

12. **Программное обеспечение, ПО**: Совокупность программ системы обработки информации и программных документов, необходимых для эксплуатации этих программ.

13**. Система управления безопасностью разработки программного обеспечения:** Часть общей системы управления, которая основана на подходе рисков при создании, внедрении, функционировании, мониторинге, анализе, поддержке и улучшении безопасности разработки ПО.

14**. Система управления конфигурациями**: Совокупность процедур и инструментальных средств (включая их документацию), используемая организацией-разработчиком ПО для разработки и поддержки конфигурации ПО в течение его жизненного цикла

15. **Статический анализ программы**: Вид работ по поиску дефектов безопасности программы, основанный на анализе исходных модулей в режиме, не предусматривающем непосредственного исполнения (функционирования) программы.

16. **Тестирование на проникновение:** Вид работ по выявлению дефектов безопасности программы, основанный на моделировании (имитации) действий потенциального нарушителя.

17**. Тестирование:** Вид работ по установлению

соответствия программы заданным требованиям и программным документам.

18. У**язвимость программы (программного обеспечения):** Дефект безопасности программы, который может быть использован для реализации недекларированных возможностей.

19**. Управление конфигурациями:** Вид деятельности, предусматривающий техническое и административное руководство и контроль, направленные на идентификацию и документирование функциональных и физических характеристик элементов конфигурации, контроль изменения этих характеристик, регистрацию и представление информации о состоянии обработки и реализации изменений, а также - верификацию соответствия установленным требованиям.

20**. Элемент конфигурации:** Объект, находящийся под управлением системы управления конфигурациями в течение разработки ПО.

4.3. Система управления безопасной разработкой программного обеспечения

1. Общие требования

Организация-разработчик ПО должна разработать, внедрить, обеспечить эксплуатацию, контролировать и совершенствовать систему управления безопасной разработкой ПО.

2. Разработка системы управления безопасной разработкой программного обеспечения

Организация-разработчик ПО должна определить и задокументировать логические и физические границы системы управления безопасной разработкой ПО с учетом:

- характеристик бизнеса организации-разработчика ПО;
- территориального размещения;
- защищаемых ресурсов;

- нормативных правовых актов, методических документов и национальных стандартов, которым должно соответствовать разрабатываемое ПО.

Организация-разработчик ПО должна:

- определить и задокументировать политику обеспечения безопасной разработки ПО, содержащую перечень целей организации в области безопасной разработки ПО и мер, направленных на достижение поставленных целей;
- определить и задокументировать подход к оценке рисков, связанных с обеспечением безопасной разработки ПО.

Меры по обеспечению безопасной разработки ПО должны быть выбраны и реализованы так, чтобы удовлетворить требованиям, определенным в процессе оценки рисков. При идентификации рисков должны быть, как минимум, быть рассмотрены угрозы, перечень которых приведен в приложении В. При выборе мер необходимо учитывать положения нормативных правовых актов, методических документов и национальных стандартов, которым должно соответствовать разрабатываемое ПО.

3. Внедрение и эксплуатация системы управления безопасной разработкой программного обеспечения

Организация-разработчик ПО должна:

- разработать план обработки рисков, определяющий соответствующие действия руководства организации-разработчика ПО, ресурсы, обязанности и приоритеты в отношении управления рисками;
- реализовать план обработки рисков для достижения поставленных целей по обеспечению безопасной разработки ПО;
- внедрить меры по обеспечению безопасной разработки ПО для достижения поставленных целей;
- определить способ измерения эффективности выбранных мер по обеспечению безопасной разработки ПО;

- управлять работой системы управления безопасной разработкой ПО;
- управлять ресурсами системы управления безопасной разработкой ПО.

4. Контроль (мониторинг) системы управления безопасной разработкой программного обеспечения

Организация должна:

- периодически пересматривать подходы к оценке рисков, анализировать остаточные риски и установленные приемлемые уровни рисков;
- проводить внутренние аудиты системы управления безопасностью разработки ПО через установленные периоды времени.

5. Совершенствование системы управления безопасной разработкой программного обеспечения

Организация должна:

- регулярно определять возможные улучшения системы управления безопасной разработкой ПО и предпринимать необходимые действия в соответствии с разделом 8;
- использовать на практике опыт по обеспечению безопасной разработки ПО, полученный как в собственной организации, так и в других организациях.

6. Требования к документации

Документация системы управления безопасной разработкой ПО должна включать в себя:

- политику обеспечения безопасной разработки ПО, содержащую перечень целей организации в области безопасной разработки ПО и мер, направленных на достижение поставленных целей;
- описание области функционирования (логические и физические границы) системы управления безопасной разработкой ПО;
- подход к оценке рисков, связанных с обеспечением безопасной разработки ПО;

- документированные свидетельства реализации мер по обеспечению безопасности разработки ПО:

1) описание методов, используемых для уникальной идентификации элементов конфигурации ПО;

2) план управления конфигурациями;

3) описание процедуры приемки элементов конфигурации ПО;

4) описание мер по обеспечению безопасной поставки ПО;

5) эксплуатационная документация на ПО;

6) журнал поэкземплярного учета дистрибутивных носителей ПО;

7) описание мер защиты инфраструктуры разработки ПО;

8) описание модели жизненного цикла ПО;

9) план обучения сотрудников организации;

10) требования безопасности, предъявляемые к разрабатываемому ПО;

11) результаты моделирования угроз безопасности информации;

12) проектная документация;

13) результаты статического и динамического анализа программы;

14) результаты инспекции исходных модулей;

15) результаты функционального и нагрузочного тестирования, тестирования на проникновение;

16) результаты отслеживания и исправления всех обнаруженных дефектов безопасности и уязвимостей программы;

17) описание инструментальных средств разработки.

Для разработки, актуализации, использования, хранения и уничтожения документов в организации должна существовать документированная процедура, определяющая действия руководства по:

- утверждению документов системы управления безопасностью разработки ПО перед их опубликованием;
- пересмотру и обновлению документов, а также повторному их утверждению;
- обеспечению идентификации внесенных изменений и текущего статуса документов;
- обеспечению наличия версий соответствующих документов в местах их использования;
- определению порядка просмотра документов и их идентификации;
- обеспечению доступа к документам авторизованным лицам, а также передачи, хранения и уничтожения в соответствии с процедурами, применимыми к степени их конфиденциальности;
- идентификации документов, созданных вне организации;
- обеспечению контроля за распространением документов;
- предотвращению непреднамеренного использования устаревших документов;
- использованию соответствующей идентификации устаревших документов в случае их дальнейшего хранения.

4.4. Обязанности руководства организации-разработчика программного обеспечения

Руководство организации-разработчика ПО должно:

- определить и утвердить политику организации в области обеспечения безопасной разработки ПО;
- довести до всех сотрудников организации, имеющих отношение к процессу разработки ПО, информацию о важности достижения целей в области обеспечения безопасной разработки ПО;

- определить сотрудников организации, ответственных за выполнение мер по обеспечению безопасной разработки ПО;
- выделить ресурсы, необходимые для разработки, внедрения, эксплуатации, контроля и совершенствования системы управления безопасной разработкой ПО;
- установить критерии принятия рисков, связанных с процессом разработки ПО;
- обеспечить проведение внутренних проверок системы управления безопасной разработкой ПО (раздел 6);
- проводить анализ системы управления безопасной разработкой ПО (раздел 4.5).

4.5. Анализ системы управления безопасной разработкой программного обеспечения

1. Руководство организации-разработчика ПО должно анализировать систему управления безопасной разработкой ПО через запланированные интервалы или при внесении существенных изменений в характеристики и цели организации по разработке ПО. Результаты анализа должны быть документированы.

2. Входные данные для анализа

Входные данные для анализа со стороны руководства организации-разработчика ПО должны включать в себя следующую информацию:

- результаты аудитов и анализа системы управления безопасной разработкой ПО;
- обратная связь заинтересованных сторон (например, органов по сертификации или испытательных лабораторий);
- методики, средства или процедуры, которые могут быть использованы в организации-разработчике ПО для

повышения эффективности системы управления безопасной разработкой ПО;

- любые изменения, которые могли повлиять на систему управления безопасной разработкой ПО.

3. Выходные данные анализа

Выходные данные анализа со стороны руководства должны включать в себя любые решения и действия, имеющие отношение к:

- повышению эффективности системы управления безопасной разработкой ПО;
- обновлению планов оценки и обработки рисков;
- изменению мер по обеспечению безопасной разработки ПО;
- потребности в ресурсах.

4.6. Совершенствование системы управления безопасной разработкой программного обеспечения

1. Организация-разработчик ПО должна проводить мероприятия по устранению причин несоответствий системы управления безопасной разработкой ПО установленным требованиям с целью предупреждения их повторного возникновения. Документированная процедура корректирующего действия должна устанавливать требования по:

- выявлению несоответствий;
- определению причин несоответствий;
- определению и реализации необходимых корректирующих действий;
- ведению записей результатов предпринятых действий.

2. Организация-разработчик ПО должна анализировать изменения перечня угроз безопасности, имеющих отношение к системе управления безопасной разработкой ПО, и устанавливать требования к предупреждающим действиям.

4.7. Меры по обеспечению безопасной разработки программного обеспечения

1. **Управление конфигурациями**

Цели:

Выполнение мер по управлению конфигурациями способствует достижению следующих целей:

- обеспечить корректность и полноту ПО перед отправкой его потребителю;
- предотвратить несанкционированную модификацию, добавление или удаление элементов конфигурации ПО.

Требования к реализации:

- Должна быть реализована процедура, позволяющая выполнять уникальную маркировку каждой версии ПО.
- Должна использоваться система управления конфигурациями, позволяющая уникально идентифицировать все элементы конфигурации ПО. Методы, используемые для уникальной идентификации элементов конфигурации ПО, должны быть документированы.

Примечание – В область действия системы управления конфигурациями, например, могут быть включены следующие элементы конфигурации ПО:

- программа;
- программная и эксплуатационная документация;
- свидетельства выполнения установленных требований
- исходные модули;
- программные и загрузочные модули;
- инструментальные средства разработки (транслятор, компилятор, ассемблер и пр.) и связанная с ними информация;
- информация, связанная с обновлениями и устранениями дефектов безопасности программы.

- Система управления конфигурациями должна осуществлять контроль доступа к элементам конфигурации ПО, при котором в них могут быть сделаны только санкционированные изменения.
- Должен быть разработан и внедрен план управления конфигурациями. План управления конфигурациями должен описывать, каким образом система управления конфигурациями используется при разработке ПО.
- Система управления конфигурациями должна поддерживать производство ПО (ассемблирование, компиляция, компоновка и т.п.) автоматизированными средствами.
- Должны использоваться процедуры приемки модифицированного или вновь созданного элемента конфигурации ПО. Используемые процедуры должны быть документированы.
- Система управления конфигурациями должна обеспечивать то, что лицо, ответственное за приемку элемента конфигурации ПО, не является разработчиком этого элемента.

Примечание – Выполнение данного требования необходимо для обеспечения выполнения принципа разделения привилегий.

- Система управления конфигурациями должна идентифицировать элементы конфигурации ПО, которые реализуют функции безопасности (при наличии в ПО функций безопасности).
- Система управления конфигурациями должна осуществлять регистрацию всех событий доступа к элементам конфигурации ПО в журнале регистрации событий. Должна регистрироваться следующая информация: тип доступа, инициатор доступа, дата и время доступа.
- Система управления конфигурациями должна предоставлять автоматизированные средства для

определения всех элементов конфигурации ПО, на которые воздействует модификация данного элемента конфигурации.

- Система управления конфигурацией должна предоставлять автоматизированные средства для резервного копирования и восстановления элементов конфигурации ПО с периодичностью, определенной планом управления конфигурацией.

2. **Безопасная поставка программного обеспечения**

Цель:

Выполнение мер по обеспечению безопасности поставки способствует достижению следующих целей:

- обеспечить точное соответствие между ПО, переданным из инфраструктуры разработки, и ПО, полученным потребителем;
- обнаружить/предотвратить модификацию ПО при его поставке потребителю;
- обеспечить потребителя эксплуатационной документацией в объеме, достаточном для правильной настройки и безопасного применения ПО.

Требования к реализации:

- Должны применяться технические и организационные меры, необходимые для обнаружения модификации ПО или любого расхождения между оригиналом и версией, полученной потребителем. Применяемые меры должны быть документированы.
- Должны применяться технические и организационные меры, необходимые для предотвращения модификации ПО или любого расхождения между оригиналом и версией, полученной потребителем. Применяемые меры должны быть документированы.
- В состав поставляемого ПО должна быть включена эксплуатационная документация в объеме, достаточном

для правильной настройки и безопасного применения ПО.

- Тиражирование дистрибутивных носителей ПО должно выполняться в соответствии с разработанными инструкциями, должен вестись поэкземплярный учет дистрибутивных носителей ПО.

3. **Защита инфраструктуры разработки программного обеспечения**

Цель:

Выполнение мер по обеспечению защиты инфраструктуры разработки ПО позволяет обеспечить конфиденциальность и целостность ПО в инфраструктуре разработки.

Требования к реализации:

Должны применяться технические и организационные меры, необходимые для обеспечения конфиденциальности и целостности объектов инфраструктуры разработки ПО. Применяемые меры должны быть документированы.

Примечание – Используемые меры направлены на обеспечение конфиденциальности и целостности следующих объектов инфраструктуры разработки ПО:

- программа (дистрибутив программы);
- программная и эксплуатационная документация;
- свидетельства выполнения установленных требований
- исходные модули;
- программные и загрузочные модули;
- инструментальные средства разработки (транслятор, компилятор, ассемблер и пр.) и связанная с ними информация;
- информация, связанная с обновлениями и устранениями дефектов безопасности программы.

Для обеспечения конфиденциальности и целостности ПО могут применяться следующие меры:

- технические: меры управления физическим доступом для предотвращения несанкционированного доступа к инфраструктуре разработки ПО;
- организационные: распространяющиеся на предоставление доступа к инфраструктуре разработки или к конкретным объектам инфраструктуры разработки, на отмену прав доступа лиц при их исключении из состава разработчиков, на передачу защищаемого материала из инфраструктуры разработки и др.;
- относящиеся к персоналу разработчиков, например, средства контроля или проверки, позволяющие установить, заслуживают ли доверия лица, принимаемые на работу;
- прочие меры безопасности, например, средства логической защиты средств разработки.

4. **Защищенное программирование**

Цель:

Использование защищенного программирования способствует достижению следующих целей:

- уменьшить количество дефектов безопасности и уязвимостей программы, поставляемой потребителю;
- устранить дефекты безопасности и уязвимости программы, выявляемые на этапе ее эксплуатации.

Требования к реализации:

- Организация-разработчик ПО должна установить и документировать модель жизненного цикла, используемую при разработке ПО.
- Должно проводиться периодическое обучение сотрудников организации-разработчика ПО с целью повышения их осведомленности в области безопасной разработки ПО. Применяемые меры должны быть документированы.

П р и м е ч а н и е – Выполнение данной меры позволит повысить осведомленность в области безопасной разработки сотрудников, связанных с разработкой ПО и, как следствие, повышение уровня безопасности разрабатываемого ПО. В программу обучения сотрудников могут входить курсы безопасного программирования, инспекции исходных модулей, тестирования на проникновение, статического и динамического анализа программы, функционального тестирования и другие. К проведению обучения сотрудников могут привлекаться сторонние организации.

- Программа обучения сотрудников должна подвергаться периодическому анализу и пересмотру при появлении существенных изменений характеристик и целей организации в области безопасности разработки ПО.
- Организацией-разработчиком ПО должны быть определены и документированы требования безопасности, предъявляемые к разрабатываемому ПО.

П р и м е ч а н и е – Определение требований безопасности к разрабатываемому ПО следует проводить в процессе анализа требований к ПО. В качестве источников для формирования требований безопасности могут, например, использоваться: перечень требований потребителя, сценарии использования ПО, требования законов, нормативных правовых актов, отраслевых стандартов и др. Например, могут быть определены следующие типы функциональных и нефункциональных (требований доверия) требования к ПО:

- требования к обеспечению идентификации и аутентификации;
- требования к обеспечению разграничения доступа;
- требования к значениям опций используемых инструментальных средств разработки;
- требования к обеспечению регистрации событий;
- требования к контролю точности, полноты и правильности данных, поступающих в программу;

- требования к заимствованным программным модулям, библиотекам и функциональным объектам;
- требования к обработке программой ошибок и исключений.

- Организацией-разработчиком ПО должно выполняться моделирование угроз с целью выявления потенциальных дефектов безопасности и уязвимостей разрабатываемой программы. Результаты моделирования угроз должны документироваться.
- Архитектура программы должна проектироваться с учетом устранения потенциальных дефектов безопасности и уязвимостей разрабатываемой программы, которые были выявлены на этапе моделирования угроз. Архитектура программы должна быть документирована.
- Организация-разработчик ПО должна идентифицировать каждое инструментальное средство, используемое при разработке ПО. Инструментальные средства и значения опций инструментальных средств, используемых при разработке ПО, должны документироваться.

Примечание – К инструментальным средствам разработки относятся, например, трансляторы, компиляторы, прикладные программы, используемые для проектирования и документирования, и пр.

Организация-разработчик ПО должна избегать использования скомпрометированных (небезопасных) конструкций языков программирования, используемых при разработке программы.

Примечание – В качестве примеров скомпрометированных (небезопасных) конструкций языков программирования (вызовов) можно привести: strcpy, strncpy, strcatm strncat, memcpy, gets.

- Организация-разработчик ПО должна проводить статический анализ программы с целью выявления

дефектов безопасности и уязвимостей программы. Результаты статического анализа программы должны документироваться.

- Организация-разработчик ПО должна проводить периодическую инспекцию исходных модулей с целью выявления дефектов безопасности и уязвимостей программы. Результаты инспекции исходных модулей программы должны документироваться.

Примечание – Инспекция исходных модулей программы выполняется сотрудниками организации-разработчика ПО или сторонних организаций, обладающими компетенцией в области выявления дефектов безопасности и уязвимостей программы на основе анализа исходных модулей программы. Поскольку инспекция является достаточно трудоемкой процедурой, то ее целесообразно проводить в отношении подмножества исходных модулей. Выбор анализируемого подмножества может осуществляться на основе установления степени важности (критичности) того или иного исходного модуля программы с точки зрения обеспечения безопасной разработки.

- Организация-разработчик ПО должна проводить тестирование (функциональное, нагрузочное) программы. Результаты тестирования должны документироваться.
- Организация-разработчик ПО должна проводить динамический анализ программы с целью выявления дефектов безопасности и уязвимостей программы. Результаты динамического анализа программы должны документироваться.
- Организация-разработчик ПО должна проводить тестирование программы на проникновение с целью выявления дефектов безопасности и уязвимостей программы. Результаты тестирования на проникновение должны документироваться.

- Организация-разработчик ПО должна реализовать процедуру, позволяющую выполнять отслеживание и исправление всех обнаруженных дефектов безопасности и уязвимостей программы в каждой версии программы. Результаты отслеживания и исправления всех обнаруженных дефектов безопасности и уязвимостей программы должны документироваться.
- Процедура устранения дефектов безопасности и уязвимостей программы должна обеспечивать прием и отработку сообщений от потребителей о дефектах безопасности и уязвимостях программы и запросов на их исправление.

ЗАКЛЮЧЕНИЕ

В данном учебном пособии:

1. Ознакомили читателей с общими вопросами создания распределенных систем, рекомендациями по архитектуре сложных приложений.
2. Были даны представления о технологиях создания распределенных приложений, поддерживаемые платформой Microsoft .NET. Показаны их особенности и взаимосвязь.
3. Были даны критерии выбора той или иной технологии при создании распределенных систем. Показали границы их применимости.
4. Было дано понятие об обеспечении безопасной разработки программного обеспечения.

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. Agha, G. Actors: a Model of Concurent Computation for Distributed Systems [Text] / G. Agha. – Cambridge MA: MIT Press, 1986.
2. Allard, J. Plug into Serious Network Programming with the Socket [Text] / J. Allard, M. Keith, T. David // PC World. – 1998. – Vol. 13.
3. Banerjee, P. Parallel Algorithms for VLSI Computer-Aided Design [Text] / P. Banerjee. – Prentice-Hall, 1994.
4. Communication Overhead for Space Science Applications on the Beowulf Parallel Workstation [Text] / D. J. Becker, T. Sterling, D. Savarese, B. Fryxell, K. Olson // Proc. High Performance and Distributed Computing. – 1995.
5. Beguelin, A. Application Level Fault Tolerance in Heterogeneous Networks of Workstations [Text] / A. Beguelin, E. Seligman, P. Stephan // Journal of Parallel and Distributed Computing: Special issue on Workstation Clusters and Network-based Computing. – 1997.
6. A decentralized control method in distributed system [Text] / J. P. Cabanel, R. D. Sazbon, A. K. Diarra, M. N. Marouane, R. Besbes // Proceeding of the First International Conference on Distributed Computing Systems. – 1979. – pp. 651-659.
7. MPVM: A Migration Transparent Version of PVM [Text] / J. Casas, D. L. Clark, R. Konuru, S. W. Otto, R. M. Prouty, J.Walpole // Computing Systems. – 1995. – Vol. 8, No. 2. – pp. 171-216.
8. Casavant, T. A Taxonomy of Scheduling in General-Purpose Distributed Computing Systems [Text] / T. Casavant, J. Kuh // IEEE Trans. Software Engineering SE-14(2), Feb. 1988, pp. 141-154.
9. David, S. Piatt. Introducing Microsoft NET [Text] / S. David // Microsoft Press. – 2000.
10. Deng, J. The Framework of Adaptive Web Search Agent [Text] / J. Deng, L. Chen // Proc. ICICS Conference. Singapore. – 2001.

11. Di Martino, B. Two Program Comprehension Tools for Automatic Parallelization [Text] / B. Di Martino, C. W. Kebler // IEEE Concurrency. – 2000. – Vol. 8, No. 1.

12. Douglas, C. Schmidt, Tim Harrison. Evaluating the Performance of 00 Network Programming Toolkits, C++ Report [Text] / C. Douglas // SIGS. – 1996. – Vol. 8, No. 7. – P. 8.

13. Enslow, P. H. What is «distributed» data processing system? [Text] / P. H. Enslow // Computer. – 1978. – Vol. 11, № 1. – pp. 13-21.

14. Clausen, E. Tecnomatix Announces First Step of Application Service Provider Initiative [Text] / E. Clausen // ASPStreet.com Application and Web services providers. – 2000.

15. Gasser, L. Social Conceptions of Knowledge and Action: DAI Foundations and Open Systems Semantics [Text] / L. Gasser // Artificial Intelligence. – 1991. – Vol. 47, № 1-3. – pp. 107-138.

16. Hockney, R. Classification and Evaluation of Parallel Computer Systems [Text] / R. Hockney // Lecture Notes in Computer Science. – 1987. – No. 295. – p. 13-25.

17. Hamilton, P. A. Measuring reliability of Computation Center Software [Text] / P. A. Hamilton, J. D. Musa // Proc. 3-th Internal. Conf. on Software, ling. – 1978. – P. 29-36.

18. MEDUSA: an experiment in distributed operating structure [Text] / J. Ousterhout et.al. // Comm. ACM. – 1980. – Vol. 23, No 2. – P. 92-105.

19. Johnson, E. Completing an MIMD Multiprocessor Taxonomy [Text] / E. Johnson // Computer Architecture News. – 1988. – Vol. 16, No 2. – P. 44-48.

20. Johnson, S. P. Automatic parallel code generation for message passing on distributed memory systems [Text] / S. P. Johnson, C. S. Ierotheou, M. Cross // Parallel Computing 22:2. – 1996. – p. 227-258.

21. Birman, K. P. Reliable communication in the presence of failures [Text] / K. P. Birman, T. A. Joseph // ACM Transactions on Computer Systems. – 1987. – Vol. 1, No 1. – P. 47-76.

22. Lea, R. Supporting Object Interaction in Heterogeneous Distributed Systems [Text] / R. Lea, J. Walpole// Computer Communications. – 1990. – Vol. 13, No. 6. – P. 365-373.

23. Столяров, М. На пути к управляемым информационным системам [Текст] / М. Столяров, И. Трифаленков // Jet Info. – 1999. – № 03.

24. Mor, H.-B. Exploiting Process Lifetime Distributions for Dynamic Load Balancing [Text] / H.-B. Mor, A. B. Downey // Proceedings of the ACM Sigmetrics Conference on Measurement and Modeling of Computer Systems. – 1996. – pp. 13-24.

25. Orly, K. Preserving Mutual Interests in High Performance Computing Clusters [Text] / K. Orly, M. Kemelmakher, I. Eshed// Journal of Parallel and Distributed Computing. – 1999. – Vol. 23, No 1. – pp. 41-48.

26. Petri, S. Load Balancing and Fault Tolerance in Workstation Clusters Migrating Groups of Communicating Processes [Text] / S. Petri, H. Langendorfer // Operating Systems Review. – 1995. –Vol. 29, No. 4. – pp. 25-36.

27. Shoham, Y. Agent Oriented Programming [Text] / Y. Shoham // Artificial Intelligence. – 1993. – Vol. 60, № 1. – pp. 51-92.

28. Welch, L. R. Engineering of Distributed Control Systems [Text] / L. R. Welch, D. K. Hammer // Parallel and Distributed Computing Practices: Special Issue. – 1998. – Vol. 1, No. 2.

29. Willebeek-LeMair, M. H. Strategies for Dynamic Load Balancing on Highly Parallel Computers [Text] / M. H. Willebeek-LeMair, A. P. Reeves // IEEE Transactions on Parallel and Distributed Systems. – 1993. – Vol. 4, No. 9. – pp. 979-993.

30. Windows Sockets API [Text] // Microsoft Systems Journal. – 1993. – Vol. 8. – P. 3550.

31. Адамов, А. З. CGI-интегрированные системы в Интернете [Электронный ресурс] / А. З. Адамов // Тез. докл. Респ. науч. конф. ИТ Баку. – 2000. – Режим доступа: http://www.taganrog.org/articles/cgi.html.

32. Адамов, А. З. JSP Технология Active Server Page (ASP) [Текст] / А. З. Адамов // Тез. докл. Респ. науч. конф. ИТ Баку, 2001, - 543с. – Режим доступа: http://www.taganrog.org/ articles/asp.html.

33. Адамов, А. З. Технология проектирования сетевых САПР [Текст] / А. З. Адамов, В. М. Глушань // Тез. докл. Межд. IEEE конф. Интеллектуальные САПР Дивноморск, 2002, с. 456-461. – Режим доступа: http://www.taganrog.org/cgi-bin/generate.cgi?content=icad&dir=arts.

34. Адамов, А. З. Дистанционное образование глазами разработчика [Текст] / А. З. Адамов // Искусственные интеллектуальные системы: тез. докл. межд. конф. – 2002. – Режим доступа: http://www.taganrog.org/cgi-bin/generate. cgi?content=depro&dir=arts.

35. Адамов, А. З. Использование ресурсов локальной сети при размещении элементов ИС методом факторизации [Текст] / А. З. Адамов // Межд. науч.-техн. конф. – 2001. – Режим доступа: http://www.taganrog.org/articles/lancad.html.

36. Адамов, А. З. Использование сокетов (Windows Sockets) [Текст] / А. З. Адамов // Респ. науч. конф. – 2001. – Режим доступа: http://www.taganrog.org/articles/socket.html.

37. Адамов, А. З., Техника использования WWW технологий для доступа к Базам Данных [Текст] / А. З. Адамов // Респ. науч. конф. – 2002. – Режим доступа: http://www.taganrog.org/cgi-bin/generate.
cgi?content=dbtry&dir=arts.

38. Кабанов, А. CALS-технологии для военной продукции [Текст] /А. Кабанов // Проблемы и решения. – 2000. – № 3.

39. Тучков, А. Создание корпоративной САПР: «как совместить желание и возможности» [Текст] / А. Тучков // САПР и графика. – 2000. – № 10.

40. Цимбал, А. Сравнительный анализ технологий CORBA и COM [Электронный ресурс] / А. Цимбал. – Режим доступа: http://city.tomsk.ru/~javadb/corba/corbacom.htm.

41. Есауленко, А. Sun раскрывает смысл Web-сервисов [Текст] / А. Есауленко // Сети. – 2002. – № 9.

42. Антонов, А. С., Эффективная адаптация последовательных программ для современных векторно-конвейерных и массивно-параллельных супер-ЭВМ [Текст] / А. С. Антонов, В. В. Воеводин // Программирование. – 1996. – № 4. – С. 37-51.

43. Аншиа, М. Л. Симфония CORBA [Текст] / М. Л. Аншиа // Открытые системы. – 1998. – Вып. 3. – С. 32-34.

44. Арапов, Д. Можно ли превратить сеть в суперкомпьютер? [Текст] / Д. Арапов // Открытые системы. – 1997. – № 4.

45. Арсеньев, Б. П. Интеграция распределенных баз данных [Текст] / Б. П. Арсеньев, С. А. Яковлев. – СПб.: Издательство «Лань», 2001. – 464 с.

46. Ахтырченко, К. В. Распределенные объектные технологии в информационных системах [Текст] / К. В. Ахтырченко, В. В. Леонтьев // СУБД. – 1997. – № 5-6. – С. 52-64.

47. Ашнина, М. Л. Предприятие как единый объект автоматизации: размышления на тему [Текст] / М. Л. Ашнина // Сети и системы связи. – 2000. – Вып. 2. – С. 23-25.

48. Баас, Р. Delphi 4. Полное руководство [Текст]: пер. с англ. / Р. Баас, М. Фервай. – Киев: BHV, 1999. – 797 с.

49. Бабушкин, М. Web-сервер [Текст] / М. Бабушкин. – СПб.: Питер, 1997.

50. Баранова, И. В. Управление предприятием на основе интегрированных средств поддержки распределённых баз данных [Текст] / И. В. Баранова // Вестник ЮжноРоссийского государственного технического университета Новочсркасского политехнического института. – 2013. –№ 1. – С. 95-108.

51. Барфилд, Э. Программирование «клиент-сервер» в локальных вычислительных сетях [Текст]: пер. с англ. / Э. Барфилд, Б. Уолтере. – М.: Филин, 1997. – 424 с.

52. Барыкин, А. Н. Механизм инновационного развития предприятия [Текст] / А. М. Барыкин // Инвестиции, инновации, экономическая безопасность: тр. секции инвестиции и экономическая безопасность. – 2004. – Вып. 8. – С. 92-98.

53. Батищев, Д. И. Методы оптимального проектирования [Текст] / Д. И. Батищев. – М.: Радио и связь, 1984. – 248 с.

54. Батищев, Д. И. Поисковые методы оптимального проектирования [Текст] / Д. И. Батищев. – М.: Совецкое радио, 1975. – 216 с.

55. Батищев, Д. И. Оптимизация в САПР [Текст] / Д. И. Батищев, Я. Е. Львович, В. Н. Фролов. – Воронеж: изд-во ВГТУ, 1997.

56. Батова, М. М. Разработка информационных систем инновационного промышленного предприятия на основе унифицированного модульного подхода [Текст] / М. М. Батова, Е. Е. Ковшов, О. С. Смирнов // Журнал об инновационной деятельности «Инновации». – 2011. – № 5(151). – С. 102-106.

57. Батова, М. М. Комплексный подход в интеллектуальном анализе данных прикладной информационной системы [Текст] / М. М. Батова, Е. Е. Ковшов, Н. Н. Митропольский // Развитие отраслевого и регионального управления. – 2011. – № 9. – С. 86-89.

58. Белянип, П. Н. О новых тенденциях развития технологии машиностроения [Текст] / П. Н. Белянип // Проблемы машиностроения и надежности машин. – 1994. – № 1. – С. 3-12.

59. Бергер, А. Б. Microsoft SQL Server 2005 Analysis Services. OLAP и многомерный анализ данных [Текст] / А. Б. Бергер. – СПб.: БХВ-Петербург, 2007. – 928 с.

60. Бершадский, А. М. Автоматизация конструкторского проектирования электронно-вычислительной и радиоэлектронной аппаратуры [Текст] / А. М. Бершадский. – Пенза: Пензенский политехнический институт, 1977. – 81 с.

61. Бершадский, А. М. Применение графов и гиперграфов для автоматизации конструкторского проектирования РЭА и ЭВА [Текст] / А. М. Бершадский. – Саратов: Изд-во Саратовского ун-та, 1983. – 120 с.

62. Бершадский, А. М. Построение систем параллельной обработки на базе кластерных технологий [Текст] / А. М. Бершадский, Л. С. Курилов, М. Н. Селиверстов // Теория и прикладные вопросы современных информационных технологий: Всероссийской конф. – 2000. – С. 40-44.

63. Бершадский, А. М. Применение кластерных технологий в САПР [Текст] / А. М. Бершадский, Л. С. Курилов, М. Н. Селиверстов // Информационные технологии. – 2001. – № 9. – С. 2-6.

64. Перспективы внедрения в химическом комплексе информационных CALS-технологий [Текст] / А. М. Бессарабов, А. Н. Афанасьев, В. П. Ефимова, Е. А. Рябенко // Химия и рынок. – 2001. – № 3. – С. 43-45.

65. Бирюков, В. Введение в CRM [Текст] / В. Бирюков, В. Дрожжипов // PC WEEK/RE. – 2001. – № 25. – С. 24-26.

66. Бландел, Р. Эффективные бизнес-коммуникации: Теория и практика в эпоху информатизации [Текст] / Р. Бландел. – СПб.: Издательство «Питер», 2000. – 380 с.

67. Богатырев, Р. Технология Curl и концепция X Internet [Текст] / Р. Богатырев // Мир ПК. – 2001. – № 9.

68. Богданов, А. В. Опыт использования PVM для параллельных вычислений [Текст] / А. В. Богданов, И. Г. Евсеев // Телематика'98: тез. докл. Междунар. конф. – 1998.

69. Борисенко, Б. В., Применение инструментальных средств обработки корпоративной информации на основе программно-аппаратных технологий [Текст] / Б. В. Борисенко, Е. Е. Ковшов // Вестник МЕТУ «Станкин». –2010. – № 3. – С. 123-129.

70. Борисов, М. UNIX-кластеры [Текст] / М. Борисов // Открытые системы. – 1995. – № 2 (10).

71. Боркус, В. CRM: мечты сбываются, популярность растет [Текст] / В. Боркус // PC WEEK/RE. – 2001. – № 45. – С. 37.

72. Интероперабельные информационные системы: архитектуры и технологии [Текст] / Д. О. Брюхов, В. И. Задорожный, Л. А. Калиниченко, М. Ю. Курошев, С. С. Шумилов // СУБД. – 1995. – № 4.

73. Интероперабельные информационные системы: архитектуры и технологии [Текст] / Д.О. Брюхов, В.И. Задорожный, Л.А. Калиниченко, М.Ю. Курошев, С.С. Шумилов // Системы Управления Базами Данных. – 1995. – № 4. – С. 96-113.

74. XML. Новые перспективы WWW [Текст]: пер с англ. / Ф. Бумфрей, О. Дирепцо, И. Дакетт и др. – М.: ДМК, 2000. – 688 с.

75. Буч, Г. Объектно-ориентированный анализ и проектирование с примерами приложений [Текст]: пер. с англ. / Г. Буч. – М.: Бином, 1998. – 560 с.

76. Буч, Г. Язык UML. Руководство пользователя [Текст]: пер. с англ. / Г. Буч, Д. Рамбо, А. Джекобсон. – М.: ДМК Пресс, 2001. – 432 с.

77. Клишин, В. Интегрированные технологии CV [Текст] / В. Клишин, В. Климов, М. Пирогова // Открытые системы. – 1997. – № 2. – С. 37-42.

78. Краюшкин, В. Система Optegra управление производственными данными [Текст] / В. Краюшкин // Открытые системы. – 1997. – № 1. – С. 67-72.

79. Олифер, В. Искусство оптимизации трафика [Текст] / В. Олифер, Н. Олифер // LAN. – 2001. – № 12.

80. Воеводин, В. В. Методы описания и классификации архитектур вычислительных систем [Текст] / В. В. Воеводин, А. П. Капитонова. – М.: Изд-во МГУ, 1994. – 79 с.

81. Многоплановая объектная модель и ее приложения [Текст] / В. Д. Волевич, В. А. Галатенко и др. // Программирование. – 1992. – № 2. – С. 24-32.

82. Вольдер, Б. С. Планирование на предприятии [Текст]: учеб. пособие / Б. С. Вольдер. – М.: МГТУ «Станкин», 1999. – 172 с.

83. Климов, В. Настоящее и будущее PDM [Текст] / В. Климов, В. Краюшкин, М. Пирогова // Открытые системы. – 2002. – № 02.

84. Генетические алгоритм разбиения графа [Текст] // Известия АН. Теория и системы управления. – 1999. – № 4. – С. 79-87.

85. Головкин, Б. А. Параллельные вычислительные системы [Текст] / Б. А. Головкин. – М.: Наука, 1980. – 520 с.

86. Городецкий, В. И. Многоагентные системы: современное состояние исследований и перспективы применения [Текст] / В. И. Городецкий // Новости искуственного интеллекта. – 1996. – № 1. – С. 44-49.

87. Горфинкель, В. Я., Экономика предприятия [Текст]: учебник для ВУЗов / В. Я. Горфинкель, Е. М. Купряков. – М.: Банки и биржи, ЮПИТИ, 1996. – 367 с.

88. Горяева, О. В. Выбор вида лицензии для программного обеспечения с открытым кодом [Текст] / О. В. Горяева, Е. Е. Ковшов, З. Ч. Нгуен // Известия вузов. Проблемы полиграфии и издательского дела. – 2009. – № 1. – С. 40-50.

89. ГОСТ Р ИСО 10303-1-99 Системы автоматизации производства и их интеграция. Представление данных об изделии и обмен этими данными. Часть 1. Общие представления и основополагающие принципы [Текст].

90. Гофман, В. Э. Работа с базами данных в Delphi [Текст] / В. Э. Гофман, А. Д. Хомонепко. – 2-е изд., перераб. – СПб: БХВ-Петербург, 2002. – 624 с.

91. Риккарди, Г. Системы баз данных. Теория и практика использования в Internet и среде Java [Текст] / Г. Риккарди. – М.: Вильяме, 2001. – 477 с.

92. Грицюк, С. Н. Математические методы и модели в экономике [Текст]: учебник / С. Н. Грицюк, Е. В. Мирзоева, В. В. Лысенко. – Ростов-на-Дону: Феникс, 2007. – 352 с.

93. Гук, М. Аппаратные средства IBM PC [Текст] / М. Гук. – СПб.: Питер, 1999. – 754 с.

94. Гурвиц, М. Безотказные сети и системы [Текст] / М. Гурвиц // Журнал сетевых решений. – 1998. – Т. 4. – № 3.

95. Волков, Д. А. Реконструкция унаследованных систем [Текст] / Д. А. Волков // Открытые системы. – 2000. – № 1-2.

96. Давыдов, В. Операционные системы для современных распределенных вычислительных систем [Текст] / В. Давыдов, А. Шелудяков // Монитор. – 1994. – № 2. – С. 6-9.

97. Дедков, А. Ф. Абстрактные типы данных в языке АТ-Паскаль [Текст] / А. Ф. Дедков. – М.: Наука, 1989. – 200 с.

98. Дейт, К. Дж. Введение в системы баз данных [Текст]: пер. с англ. / К. Дж. Дейт. – 8-е изд. – М.: Изд. дом «Вильяме», 2005.

99. Джамса, К. Программирование для Internet в среде Windows [Текст]: пер с англ. / К. Джамса, К. Коуп. – СПб.: Питер, 1996. – 688 с.

100. Борк, Д. Будущее сетевых приложений [Текст] / Д. Борк // Computerworld. – 2001. – № 37.

101. Борк, Д. Споры вокруг SOAP [Текст] / Д. Борк // Computerworld. – 2000. – № 35.

102. Джонс, М. Т. Программирование искусственного интеллекта в приложениях [Текст]: пер. с англ. / М. Т. Джонс. – М.: ДМК Пресс, 2004. – 312 с.

103. Димитриев, Ю. К. Анализ самодиагностических свойств структур распределенных живучих вычислительных систем [Текст] / Ю. К. Димитриев // Автометрия. – 1997. – № 5. – С. 71-84.

104. Димитриев, Ю. К. Самодиагностика модульных вычислительных систем [Текст] / Ю. К. Димитриев. – Новосибирск: Наука, 1993. – 292 с.

105. Димитриев, Ю. К. Вычислительные системы из мини-ЭВМ [Текст] / Ю. К. Димитриев, В. Г. Хорошевский. – М.: Радио и связь, 1982. – 304 с.

106. Дмитров, В. И. CALS как основа для проектирования виртуальных предприятий [Текст] / В. И. Дмитров // Автоматизация проектирования. – 1997. – № 5.

107. Дмитров, В. И. Опыт внедрения CALS за рубежом [Текст] / В. И. Дмитров // Автоматизация проектирования. – 1997. – № 1.

108. Дмитров, В. И. CALS-стандарты [Текст] / В. И. Дмитров, Ю. М. Макаренков // Автоматизация проектирования. – 1997. – № 2, 3, 4.

109. Дубова, Н. COM или CORBA? Вот в чем вопрос [Текст] / Н. Дубова // Открытые системы. – 1999. – № 3.

110. Дубова, Н. Интегрированные системы управления распределенной корпорацией [Текст] / Н. Дубова // Открытые системы. 1998. – Вып. 1. – С. 13-18.

111. Дуг, А. Gigabit Ethernet покоряет в города [Текст] / А. Дуг // LAN. – 2001. – № 11.

112. Дунаев, С. INTRANET-технологии [Текст] / С. Дунаев. – М.: Диалог-МИФИ, 1997. – 288 с.

113. Дунаев, С. А. INTRANET- технологии. WebDBC. CGI. CORBA 2.0. Netscape. Suite. Borland. IntraBuilder. Java. JavaScript. Live Wire [Текст] / С. А. Дунаев. – М.: Диалог-МИФИ, 1997. – 228 с.

114. Гринфилд, Д. Анатомия самой новой сети общего доступа [Текст] / Д. Гринфилд // LAN. – 2001. – № 12.

115. Гринфилд, Д. Глобальный телекоммуникационный сервис глазами пользователей [Текст] / Д. Гринфилд // LAN. – 2002. – № 02.

116. Левшаков, Е. Особенности использования технологии CORBA в VisiBroker for Delphi на примере mcsa.ru [Текст] / Е. Левшаков // Корпоративные системы. – 2001.

117. Е. Судов. Информационная поддержка жизненного цикла продукта [Текст] / Е. Судов //PC Week. – 1998. – № 45.

118. Евреинов, Э. В. Однородные вычислительные системы [Текст] / Э. В. Евреинов, В. Г. Хорошевский. – Новосибирск: Наука, 1978. – 320 с.

119. Егорова, И. Е. Интеллектуальная система на основе генетических алгоритмов для оптимизации состава организационных структур в области разработки и внедрения инноваций [Текст]: автореф. дис. . канд. эк. наук. / И. Е. Егорова. – Волгоград: РПК «Политехник» ВЕТУ, 2006. – 26 с.

120. Каминский, И. Маршрутизаторы доступа «все-в-одном» [Текст] / И. Каминский // LAN. – 2001. – № 11.

121. Методология разработки распределенных приложений на основе модели оболочки [Текст] / В. П. Иванников и др. // Вопросы кибернетики. Приложения системного программирования. – 1997. – Вып. 3.

122. Дубравский, И. Беспроводная (не)безопасность [Текст] / И. Дубравский // LAN. – 2002. – № 06.

123. Искуственный интеллект: системы общения и экспертные системы [Текст]: справочник / под ред. Э. В. Попова. – М.: Радио и Связь, 1990. – 464 с.

124. Автоматизация проектирования вычислительных структур [Текст] / А. В. Каляев, А. Н. Мелихов, В. М. Курейчик, В. Ф. Гузик, В. А. Калашников. – Ростов: Изд-во Ростовского ун-та, 1983. – 224 с.

125. Калянов, Г. Н. Теория и практика реорганизации бизнес-процессов [Текст] / Г. Н. Калянов. – М.: СИНТЕГ, 2000.

126. Энди, К. М. Быстрая и качественная разработка программного обеспечения [Текст]: пер. с англ. / К. М. Энди, Х. Ден. – М.: Издательский дом «Вильяме», 2003. – 400 с.

127. Карпов, Ю. Е. Теория автоматов [Текст] / Ю. Е. Карпов. – СПб: Питер, 2003. – 208 с.

128. Кейслер, С. Проектирование операционных систем для малых ЭВМ [Текст]: пер. с англ. / С. Кейслер. – М.: Мир, 1986. – 680 с.

129. Кирсанов, Д. Веб-дизайн [Текст] / Д. Кирсанов. – СПб: Символ-Плюс, 1999.

130. Киселев, М. Средства добычи знаний в бизнесе и финансах [Текст] / М. Киселев, Б. Соломатин // Открытые системы. – 1997. – № 4. – С. 41-44.

131. Управление проектом по созданию интернет-сайта [Текст] / А. Ковалев, И. Кудрюмов и др. – М.: Альпина Паблишер, 2001. – 337 с.

132. Ковшов, Е. К. Методы оценки и повышения надёжности информационных сред промышленного предприятия [Текст] / Е. К. Ковшов, О. С. Смирнов // Динамика сложных систем-XXI век. – 2012. – № 2. – С. 112-116.

133. Козлова, Е. А. Некоторые моменты анализа экономических рисков на микроуровне [Текст] / Е. А. Козлова // Экономика, управление и инвестиции: сборник научных трудов. – 2004. – Вып. 5.

134. Козлова, Е. А. Теоретические и методические основы исследования экономических рисков [Текст] / Е. А. Козлова, И. Г. Шепелев // Стратегический управленческий анализ. – 2007. – № 3.

135. Кознов, Д. В. Проблемы разработки компонентного программного обеспечения. Объектно-ориентированное визуальное моделирование [Текст] / Д. В. Кознов; под ред. проф. А. М. Терехова. – СПб: Издательство С.-Петербургского университета, 1999. – С. 86-100.

136. Компьютерно-интегрированные производства и CALS-технологии в машиностроении [Текст] / под ред. Б. И. Черпакова. – М.: ВИМИ, 1999. – 512 с.

137. Коналлен, Д. Разработка Wcb-приложений с использованием [Текст]: пер. с англ. / Д. Коналлен. – М.: Издательский дом «Вильяме», 2001. – 288 с.

138. Евченко, К. Как выбрать CAD-систему [Текст] / К. Евченко // САПР и графика. – 2002. – № 5.

139. Корнеев, В. В. Параллельные вычислительные системы [Текст] / В. В. Корнеев. – М.: Нолидж, 1999. – 320 с.

140. Базы данных. Интеллектуальная обработка информации [Текст]: Т. 1 / В. В. Корнеев, А. Ф. Гареев, С .В. Васютин, В. В. Райх. – М.: «Нолидж», 2000. – 352 с.

141. Базы данных. Интеллектуальная обработка информации [Текст]: Т. 2 / В. В. Корнеев, А. Ф. Гареев, С. В. Васютин, В. В. Райх. – М.: «Нолидж», 2000. – 150 с.

142. Корячко, В. П. Теоретические основы САПР [Текст] / В. П. Корячко, В. М. Курейчик, И. П. Норенков. – М.: Энергоатомиздат, 1987. – 400 с.

143. Косихин, Б. CRM практика для укрепления связи [Текст] / Б. Косихин // PC WEEK/RE. – 2001. – № 45. – С. 37-41.

144. Костенко, В. А. Метод автоматизированного синтеза параллельных программ для вычислительных систем с M1ME) архитектурой [Текст] / В. А. Костенко // Программирование. – 1993. – № 1. – С. 43-57.

145. Коффи, П. e-CKM: покупатель становится королем? [Текст] / П. Коффи // PC WEEK/RE. – 2001. – № 22. – С. 29.

146. Кристиансен, Т. Perl. Библиотека программиста [Текст] / Т. Кристиансен, Н. Торкингтон. – СПб.: Питер, 2000. – 734 с.

147. Кузнецова, О. Б. Расчет экономической эффективности от внедрения ИТ-проектов [Текст] / О. Б. Кузнецова, С. Л. Шиманский. – Мурманск: МГТУ, 2012. – 31 с.

148. Кузьминский, М. OpenMP: средства распараллеливания для многопроцессорных систем [Текст] / М. Кузьминский // Открытые системы. – 1998. – № 3.

149. Курейчик, В. М., Генетические алгоритмы и их применение [Текст] / В. М. Курейчик. – Таганрог: ТРТУ, 2002. – С. 128-175.

150. Курейчик, В. М., Генетические алгоритмы и их применение [Текст] / В. М. Курейчик. – Таганрог: ТРТУ, 2002. – С. 128-149.

151. Курейчик, В. М., Генетические Алгоритмы [Текст]: монография / В. М. Курейчик. – Таганрог: ТРТУ, 1998. – 242 с.

152. Курейчик, В. М. Методы управления в ИС принятия решений на основе эволюционного моделирования [Текст] / В. М. Курейчик, В. В. Курейчик // Искусственный интеллект в XXI веке: тр. межд. конф. – 2002. – С. 488-499.

153. Курилов Л. С. Организация ядра оболочки распределенных вычислений [Текст] / Л. С. Курилов // Интеллектуальные САПР-2000: Тез. докл. XV Междунар. науч.-техн. конф. – 2000.

154. Курилов, Л. С. Распределенная САПР: преимущества и перспективы [Текст] / Л. С. Курилов // Высокие технологии в региональной информатике: тез. докл. Всеросс. совещания-семинара. – 1998. – Ч. I. – С. 46-47.

155. Курилов, Л. С. Технология межмашинного взаимодействия в полностью распределенной среде [Текст] / Л. С. Курилов // Интеллектуальные информационные системы: труды Всеросс. конф. – 1999. – С. 60-61.

156. Курилов, Л. С. Гетерогенная объектно-ориентированная среда распределенных вычислений [Текст] / Л. С. Курилов // Информационные системы и технологии: материалы Межд. конф. – 2000. – Т. 3. – С. 462.

157. Курилов, Л. С. Объектная модель распределенных вычислений на основе поведенческого подхода [Текст] / Л. С. Курилов // Новые информационные технологии обучения в региональной инфраструктуре: тез. докл. II межрегион. науч.-методич. конф. – 1999. – С. 47-48.

158. Курилов Л. С. Парадигма распределенного программирования как средство развития нетрадиционного мышления [Текст] / Л. С. Курилов // Университетское образование: материалы III Межд. науч.-методич. конф. – 1999. – С. 131-133.

159. Курилов, Л. С. Применение технологий параллельного и распределенного программирования в учебном процессе [Текст] / Л. С. Курилов // Университетское образование: сб. материалов IV Межд. науч.-методич. конф. – 2000. – Ч. I. – С. 75-77.

160. Курилов, Л. С. Проблема авто динамического синтеза оптимальной структуры кластера [Текст] / Л. С. Курилов // Информационные технологии и системы в образовании, науке, бизнесе: сб. материалов Межд. науч.-техн. конф. – 1999. – С. 32-34.

161. Курилов, Л. С. Программная оболочка для проведения практических занятий в ДО [Текст] / Л. С. Курилов // Новые компьютерные технологии обучения в региональной инфраструктуре: тез. докл. I межрегион. науч.-методич. конф. – 1998. – С. 74-75.

162. Куцевич, Н. Интеграция АСУП и ЛСУТП [Текст] / Н. Куцевич // Открытые системы. – 2000. – Вып. 9. – С. 30-34.

163. Лебедев, Б. К. Методы поисковой адаптации в задачах автоматизированного проектирования СБИС [Текст] / Б. К. Лебедев. – Таганрог: ТРТУ, 2000. – С. 41-62

164. Черняк, Л. Серверы корпоративных приложений — ОС для корпоративных информационных систем? [Текст] / Л. Черняк // Открытые системы. – 2001. – № 01.

165. Лясковская, Е. А. Теоретические и методологические основы управления инновационным развитием предприятия по показателям устойчивости [Текст] / Е. А. Лясковская // Вестник университета (ГУУ). – 2009. – № 13.

166. Аншина, М. Симфония CORBA [Текст] / М. Аншина // Открытые системы. – 1998. – № 03.

167. Ильина, М. «Правильная» автоматизация для России [Текст] / М. Ильина // Computerworld. – 2001. – № 15,16.

168. Мартин, Дж. Вычислительные сети и распределенная обработка данных: программное обеспечение, методы и архитектура [Текст]: пер. с англ. / Дж. Мартин // Финансы и статистика. – 1985. – Вып. 1, 2.

169. Матросов, А. HTML 4.0: Полное руководство [Текст] / А. Матросов, А. Сергеев. – М.: BHV, 2000.

170. Мелихов, А. Н. Применение графов для проектирования дискретных устройств [Текст] / А. Н. Мелихов, Л. С. Берштейн, В. М. Курейчик. – М.: Наука, 1974. – 304 с.

171. Егоров, М. Концепция создания иерархической интегрированной САПР предприятия в едином информационном пространстве корпорации [Текст] / М. Егоров // САПР и графика. – 2001. – № 11.

172. Зырянов, М. Sterling движется от системной интеграции к консалтингу [Текст] / М. Зырянов // Computerworld. – 2001. – № 26, 27.

173. Монахова, Г. СКМ и подсолнухи близнецы-братья [Текст] / Г. Монахова // PC \VEEK7RE. – 2001. – № 40. – С. 28.

174. Морган, М. Java 2: Руководство разработчика [Текст] / М. Морган. – М.: Вильямс, 2000.

175. Морозов, К. К., Методы разбиения схем РЭА на конструктивно законченные части [Текст] / К. К. Морозов. –М.: Радио и связь, 1978.

176. Морозов, К. К. Автоматизированное проектирование конструкций радиоэлектронной аппаратуры [Текст] / К. К. Морозов, В. Г. Одиноков, В. М. Курейчик. – М.: Радио и связь, 1983. – 280 с.

177. Москалев, Л. Л. CORBA в промышленных приложениях [Текст] / Л. Л. Москалев. // Мир компьютерной автоматизации. – 2001. – Вып. 5. – С. 60-62.

178. Елманова, Н. Использование CORBA для организации распределенных вычислений [Текст] / Н. Елманова // Компьютер Пресс. – 1999. – № 04.

179. Пирогова, Н. Управление производственными данными на базе Web [Текст] / Н. Пирогова // Открытые системы. – 1998. – № 06.

180. Немет, Р. Unix. Руководство системного администратора [Текст] / Р. Немет, Г. Снайдер. – Киев: BHV, 2000.

181. Норенков, И. П. Введение в автоматизированное проектирование технических устройств и систем [Текст] / И. П. Норенков. – М.: Высшая школа, 1980. – 311с.

182. Норенков, И. П. Основы теории и проектирования САПР [Текст] / И. П. Норенков, В. Б. Маничев. – М.: Высшая школа, 1990. – 335 с.

183. Норенков, И. П. Системы автоматизированного проектирования электронной и вычислительной аппаратуры

[Текст] / И. П. Норенков, В. Б. Маничев. – М.: Высшая школа, 1983. – 272 с.

184. Обсрг, Р. Технология COM+. Основы и программирование [Текст]: пер. с англ. / Р. Обсрг. – М.: Вильяме, 2000. – 480 с.

185. Овсянников, М. CALS повышает конкурентоспособность изделий [Текст] / М. Овсянников, С. Сумароков // PC Week. – 2001. – Вып. 11. – С. 45-48.

186. Ойхман, Т. Т. Реинжиниринг бизнеса: Реинжиниринг организаций и информационные технологии [Текст] / Т. Т. Ойхман, Э. В. Попов. – М.: Финансы и статистика, 1997. – 336 с.

187. Окулсссский, В. Д. К оценке эффективности реинжиниринга бизнес-процессов [Текст] / В. А. Окулсссский, А. И. Левин // Информационные технологии в проектировании и производстве. – 2000. – № 2. – С. 25-28.

188. Орлик, С. В ожидании CORBA 3.0 [Текст] / С. Орлик // Открытые системы. – 1999. – № 2.

189. Отоцкий, Л. Тернистый путь к современной технологии управления [Текст] / Л. Отоцкий, А. Савин // Открытые системы. – 1998. – № 2.

190. Шуманов, П. В. Delphi 4 Руководство разработки баз данных [Текст] / П. В. Шуманов, В. В. Фаронов. – М.: Нолидж, 1999. – 491 с.

191. Брук, П. Web-центричное взаимодействие [Текст] / П. Брук // Открытые системы. – 2002. – № 2.

192. Погребинский, А. Сравнительный анализ CAD/CAM-систем [Текст] / А. Погребинский, А. Павлов // САПР и графика. – 2000. – № 8.

193. Пономарев, В. М. Системное проектирование интегрированных производственных комплексов [Текст] / В. М. Пономарев. – Ленинград: Машиностроение, 1986. – 318 с.

194. Статические и динамические экспертные системы [Текст] / Э. В. Попов и др. – М.: Финансы и статистика, 1996. – 320 с.

195. Причард, Д. COM и CORBA [Текст]: пер. с англ. / Д. Причард. – М.: ЛОРИ, 2001. – 372 с.

196. Программирование на параллельных вычислительных системах [Текст]: пер. с англ. / Р. Бэбб, Дж. Мак-Гроу, Т. Акселрод и др.; под ред. Р. Бэбба. – М.: Мир, 1991. – 376 с.

197. Птипьер, К. Синхронный C++ для интерактивных приложений [Текст] / К. Птипьер // Открытые системы. – 1999. – № 3.

198. Равашвами, Р. Оптимизация работы приложений в глобальной сети [Текст] / Р. Равашвами // LAN. – 2000. – № 1.

199. Ратшиллер, Р. PHP. Разработка Web-приложений [Текст] / Р. Ратшиллер. – СПб.: Питер, 2001.

200. Орфали, Р. Основы CORBA [Текст] / Р. Орфали, Д. Харки, Д. Эдварде. – М.: МАЛИП, 1999.

201. Роджерсон, Д. Основы COM [Текст]: пер. с англ. / Д. Роджерсон. – М.: Русская редакция, 2000. – 400 с.

202. Рутковская, Д. Нейронные сети, генетические алгоритмы и нечеткие системы [Текст]: пер. с польск. / Д. Рутковская, М. Пилинъский, Л. Рутковский. – М.: Горячая линия–Телеком, 2004. – 452 с.

203. Жанг, Р. Сетевые услуги нового поколения [Текст] / Р. Жанг // LAN. – 2001. – № 12.

204. Семенов, Ю. А. Протоколы и ресурсы Internet [Текст] / Ю. А. Семенов. – М.: Радио и связь, 1996. – 320 с.

205. Марьин, С. Компьютерные технологии для проектирования и производства сложных изделий машиностроения [Текст] / С. Марьин // САПР и графика. – 2000. – № 7.

206. Системы параллельной обработки [Текст]: пер. с англ. / Ж.-Л. Баер, Р. Барлоу, М. Вудворд и др.; под ред. Д. Ивенса. – М.: Мир, 1985. – 416 с.

207. Слама, Д. Корпоративные системы на основе CORBA [Текст]: пер. с англ. / Д. Слама, Д. Гарбис, П. Рассел. – М.: Вильяме, 2000. – 368 с.

208. Смелянский, Р. Л. Применение темпоральной логики для спецификации поведения программных систем [Текст] / Р. Л. Смелянский // Программирование. – 1993. – № 1. – С. 3-28.

209. Автоматизация доступа к информационным ресурсам распределенных баз данных современного производства [Текст] / Е. Т. Стамировски, В. К. Шемелин, Е. Е. Ковшов, М. Г. Косов // Мехатроника, Автоматизация, Управление. – 2004. – № 1. – С. 10-18.

210. Столбовский Д. Н. Основы разработки Web-приложений на ASP.NET [Текст] / Д. Н. Столбовский. – М.: Бином. Лаборатория знаний, 2009. – 304 с.

211. Концепция развития CALS-технологий в промышленности России [Текст] / Е. В. Суворов, А. И. Левин, А. П. Давыдов, В. В. Барабанов. – М.: НИЦ CALS-технологий «Прикладная логистика», 2002. – 153 с.

212. Сырков, Б. Ю. Программное обеспечение мультитранспьютерных систем [Текст] / Б. Ю. Сырков, С. В. Матвеев. – М.: ДИАЛОГ-МИФИ, 1992. – 224 с.

213. Тарасов, В. Б. От многоагентных систем к интеллектуальным организациям [Текст] / В. Б. Тарасов. – М.: УРСС, 2002. – С. 101-151.

214. Теория и методы автоматизированного проектирования вычислительных систем [Текст] / под ред. Брейера. – М.: Мир, 1977. – 282 с.

215. Тридцать три причины, почему клиенты должны быть тонкими [Текст] // Jetlnfo. – 2000. – № 6.

216. Тудер, И. Ю. Коллективный анализ предметной области [Текст] / И. Ю. Тудер // Банковские технологии. – 2001. – № 5. – С. 32-38.

217. Тюрин, Ю. Н. Статистический анализ данных на компьютере [Текст] / Ю. Н. Тюрин, А. А. Макаров. – М.: Инфра-М, 1998. – 528 с.

218. Фролов, В. Н. Автоматизированное проектирование технологических процессов и систем

производства РЭС [Текст]: учеб. пособие / В. Н. Фролов, Я. Е. Львович, Н. П. Меткин. – М.: Высшая школа, 1991. – 463 с.

219. Хаббард, Дж. Автоматизированное проектирование баз данных [Текст] / Дж. Хаббард. – М.: Мир, 1984.

220. Халсалл, Ф. Передача данных, сети компьютеров и взаимосвязь открытых систем [Текст]: пер. с англ. / Ф. Халсалл. – М.: Радио и связь, 1995. – 408 с.

221. Хаусли, Т. Системы передачи и телеобработки данных [Текст]: пер. с англ. / Т. Хаусли. – М.: Радио и связь, 1994. – 456 с.

222. Холл, М. Сервлеты и JavaServer Pages [Текст] / М. Холл. – СПб.: Питер, 2001.

223. Цумаков, П. В. Delphi 4. Руководство разработчика баз данных [Текст] / П. В. Цумаков, В. В. Фаронов. – М.: Нолидж, 1999. – 557 с.

224. Чан, Т. Системное программирование на C++ для UNIX [Текст]: пер. с англ. / Т. Чан. – Киев: BHV, 1999. – 592 с.

225. Чеботарев, В. Моделирование корпоративного портала знаний [Текст] / В. Чеботарев // PC Week /RE. – 2001. – № 14.

226. Черненький, В. М. Имитационное моделирование [Текст]: практич. пособие / В. М. Черненький; под ред. А. В. Петрова. – М.: Высшая школа, 1990.

227. Шапошников, И. В. Web-ссрвисы Microsoft .Net [Текст] / И. В. Шапошников. – СПб.: БХВ-ГГстербург, 2002. – 336 с.

228. Швецов, И. Е. ТАО технология активных объектов для разработки мультиагентных систем [Текст] / И. Е. Швецов, Т. В. Нестеренго, С. А. Старовин // Информационные технологии и вычислительные системы. – 1998. – № 1. – С. 35-43.

229. Шемелин, В. К. Роль WEB-сервисов в разработке приложений в распределенной среде [Текст] / В. К. Шемелин, М. Н. Ульяпычев // Объединенный научный журнал. – 2004. – № 29. – С. 73-74.

230. Шемелин, В. К. Анализ технологии WEB-сервисов .Net с целью построения сервера приложений в распределенной архитектуре [Текст] / В. К. Шемелин, М. Н. Ульяпычев // Объединенный научный журнал. – 2004. – № 29. – С. 75-77.

231. Широков, С. В. Модели и методы управления вычислениями в технических системах на основе применения активных баз данных [Текст]: автореф. дисс. … канд. техн. наук / С. В. Широков. – Пенза, 1997. – 404 с.

232. Дорнан, Э. Ethernet выходит в глобальные сети [Текст] / Э. Дорнан // LAN. – 2000. – № 11.

233. Дорнан, Э. Есть ли у ATM перспектива? [Текст] / Э. Дорнан // LAN. – 2001. – № 03.

234. Энкарначчо, Ж. Автоматизированное проектирование. Основные понятия и архитектура систем [Текст]: пер. с англ. / Ж. Энкарначчо, Э. Шлехтендаль. – М.: Радио и связь, 1986. – 288 с.

235. Пуха, Ю. CORBA/IIOP и Java RMI. Основные возможности в сравнении [Текст] / Ю. Пуха // СУБД. – 1997. – № 04.

236. Юткин, А. Объектные технологии в распределенных системах [Текст] / А. Юткин // Открытые системы. – 1995. – № 3 (11).

СОДЕРЖАНИЕ

ВВЕДЕНИЕ 3
1. ВВЕДЕНИЕ В РАСПРЕДЕЛЕНИЕ СИСТЕМЫ..... 4
1.1. Понятие распределенной системы 4
1.2. Определение распределенной системы. Программные компоненты..... 9
1.3. Требования к распределенным системам 11
1.4. Понятие промежуточной среды..... 15
2. ВЗАИМОДЕЙСТВИЕ КОМПОНЕНТ РАСПРЕДЕЛЕННОЙ СИСТЕМЫ 19
2.1. Модели взаимодействия компонент распределенной системы 19
2.2. Обмен сообщениями..... 20
2.3. Дальний вызов процедур..... 22
2.4. Использование удаленных объектов..... 24
2.5. Распределенные события 32
2.6. Распределенные транзакции 33
3. ОБЕСПЕЧЕНИЕ ФУНКЦИОНАЛЬНОЙ БЕЗОПАСНОСТИ РАСПРЕДЕЛЕННЫХ ПРИЛОЖЕНИЙ 36
3.1. Проблемы обеспечения функциональной безопасности программных средств 36
3.2. Основные понятия и факторы, определяющие безопасность программных средств..... 47
3.3. Характеристики среды, для которой должна обеспечиваться функциональная безопасность программных средств 72
3.4. Ресурсы для обеспечения функциональной безопасности программных средств 85

4. ОБЕСПЕЧЕНИЕ БЕЗОПАСНОЙ РАЗРАБОТКИ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ 100
4.1. Нормативно-правовые основы обеспечения безопасной разработки программного обеспечения 100
4.2. Термины и определения 106
4.3. Система управления безопасной разработкой программного обеспечения 108
4.4. Обязанности руководства организации-разработчика программного обеспечения 112
4.5. Анализ системы управления безопасной разработкой программного обеспечения 113
4.6. Совершенствование системы управления безопасной разработкой программного обеспечения 114
4.7. Меры по обеспечению безопасной разработки программного обеспечения 115
ЗАКЛЮЧЕНИЕ 124
БИБЛИОГРАФИЧЕСКИЙ СПИСОК 125

Учебное издание

Карпеев Дмитрий Олегович
Куликов Сергей Сергеевич

ТЕХНОЛОГИЯ ПОСТРОЕНИЯ ЗАЩИЩЕННЫХ РАСПРЕДЕЛЕННЫХ ПРИЛОЖЕНИЙ

В авторской редакции

Подписано к изданию 27.08.2015.

Объем данных 1,26 Мб.

ФГБОУ ВПО «Воронежский государственный технический университет»
394026 Воронеж, Московский просп., 14