### **Модель факторов, определяющих надежность программных средств**

При любом виде деятельности людям свойственно непредумышленно ошибаться, результаты чего проявляются в процессе создания или применения изделий или систем. В общем случае под ошибкой подразумевается дефект, погрешность или не умышленное искажение объекта или процесса. При этом предполагается, что известно правильное, эталонное состояние объекта, по отношению к которому может быть определено наличие отклонения – дефекта или ошибки. Для систематической, координированной борьбы с ними необходимы исследования факторов, влияющих на надежность ПС со стороны случайных, существующих и потенциально возможных дефектов в конкретных программах. Это позволит целенаправленно разрабатывать комплексы методов и средств обеспечения надежности сложных ПС различного назначения при реально достижимом снижении уровня дефектов проектирования.

При строго фиксированных исходных данных программы исполняются по определенным маршрутам и выдают совершенно определенные результаты. Многочисленные варианты исполнения программ при разнообразных исходных данных представляются для внешнего наблюдателя как случайные. В связи с этим дефекты функционирования программных средств, не имеющие злоумышленных источников, проявляются внешне как случайные, имеют разную природу и последствия. В частности, они могут приводить к последствиям, соответствующим нарушениям работоспособности, и к отказам при использовании ПС.

Объектами уязвимости (см. таблицу 1), влияющими на надежность ПС, являются:

– динамический **вычислительный процесс** обработки данных, автоматизированной подготовки решений и выработки управляющих воздействий;

– **информация**, накопленная **в базах данных**, отражающая объекты внешней среды, и процессы ее обработки;

– **объектный код программ**, исполняемых вычислительными средствами в процессе функционирования ПС;

– **информация**, выдаваемая **потребителям** и на исполнительные механизмы, являющаяся результатом обработки исходных данных и информации, накопленной в базе данных.

На эти объекты воздействуют различные дестабилизирующие факторы, которые можно разделить на внутренние, присущие самим объектам уязвимости, и внешние, обусловленные средой, в которой эти объекты функционируют.

Внутренними источниками угроз надежности функционирования сложных ПС можно считать следующие дефекты программ:

– системные **ошибки проектирования при постановке** целей и **задач** создания ПС, при формулировке требований к функциям и характеристикам решения задач, определении условий и параметров внешней среды, в которой предстоит применять ПС;

– **алгоритмические ошибки** разработки при непосредственной спецификации функций программных средств, при определении структуры и взаимодействия компонентов комплексов программ, а также при использовании информации баз данных;

– **ошибки программирования** в текстах программ и описаниях данных, а также в исходной и результирующей документации на компоненты и ПС в целом;

– **недостаточную** эффективность используемых методов и средств оперативной **защиты** программ и данных от сбоев и отказов и обеспечения надежности функционирования ПС в условиях случайных негативных воздействий.

Внешними дестабилизирующими факторами, отражающимися на надежности функционирования перечисленных объектов уязвимости в ПС, являются:

– **ошибки** оперативного и обслуживающего **персонала** в процессе эксплуатации ПС;

– **искажения** **в каналах** телекоммуникации **информации**, поступающей от внешних источников и передаваемой потребителям, а также недопустимые для конкретной информационной системы характеристики потоков внешней информации;

– **сбои и отказы в аппаратуре** вычислительных средств;

– **изменения** состава и **конфигурации** комплекса взаимодействующей аппаратуры информационной **системы** за пределы, проверенные при испытаниях или сертификации и отраженные в эксплуатационной документации.

**Таблица 1. – Объекты уязвимости и последствия нарушения надежности**

|  |  |
| --- | --- |
| Объекты уязвимости | Последствия нарушения надежности |
| Вычислительный процесс | Разрушение вычислительного процесса |
| Информация БД | Разрушение информации БД |
| Объектный код программ | Разрушение текста программ |
| Информация для потребителей | Разрушение информации для потребителей |

Полное устранение перечисленных негативных воздействий и дефектов, отражающихся на надежности функционирования сложных ПС, принципиально невозможно. Проблема состоит в выявлении факторов, от которых они зависят, создании методов и средств уменьшения их влияния на надежность ПС, а также в эффективном распределении ресурсов на защиту для обеспечения необходимой надежности комплекса программ, равноправного при их реальных воздействиях.

Современные достижения микроэлектроники снизили влияние сбоев и отказов вычислительных средств на надежность функционирования ПС. Однако ошибки персонала, искажения данных в каналах телекоммуникации, а также случайные (при отказах части аппаратуры) и необходимые изменения конфигурации вычислительных средств остаются существенными внешними угрозами надежности ПС. Негативное влияние этих факторов может быть значительно снижено соответствующими методами и средствами защиты и восстановления программ и данных. Внешние дестабилизирующие факторы имеют различную природу и широкий спектр характеристик, которые представлены во многих публикациях. Поэтому далее сделан акцент на внутренних дестабилизирующих факторах, различного рода дефектах и ошибках проектирования и эксплуатации, оказывающих большее влияние на надежность функционирования 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 и полноты исходных данных для проектирования спецификаций сложных ПС. 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 проектирования. Однако установить ошибочность исходных данных и спецификаций еще труднее, чем обнаружить ошибки в созданных программах и данных, так как принципиально отсутствуют формализованные данные, которые можно использовать как эталонные, и их заменяют неформализованные представления заказчиков и разработчиков.

Степень влияния всех внутренних дестабилизирующих факторов, а также некоторых внешних угроз на надежность ПС определяется в наибольшей степени качеством технологий проектирования, разработки, сопровождения и документирования ПС и их основных компонентов. При ограниченных ресурсах на разработку ПС для достижения заданных требований по надежности необходимо управление обеспечением качества в течение всего цикла создания программ и данных. Такое управление предполагает высокую дисциплину и проектировочную культуру всего коллектива специалистов, использование им методик, типовых нормативных документов и средств автоматизации разработки.

Кроме того, обеспечение качества ПС предполагает формализацию и сертификацию технологии их разработки, а также выделение в специальный процесс поэтапного измерения и анализа текущего качества создаваемых и применяемых компонентов. Попытки создания сложных, распределенных ПС на базе мультипроцессорных ЭВМ и концепции клиент-сервер без использования эффективных технологий и средств автоматизации проектирования связаны с высоким риском полного провала проектов вследствие трудностей обеспечения необходимой надежности функционирования таких систем.

### **Методы обеспечения надежности программных средств**

В современных автоматизированных технологиях создания и развития сложных ПС с позиции обеспечения их необходимой и заданной надежности можно выделить методы и средства, позволяющие:

– создавать программные модули и функциональные компоненты высокого, гарантированного качества;

– **предотвращать дефекты проектирования** **за счет** эффективных технологий и **средств автоматизации** обеспечения всего жизненного цикла комплексов программ и баз данных;

– обнаруживать и устранять различные дефекты и ошибки проектирования, разработки и сопровождения программ путем **систематического тестирования** на всех этапах жизненного цикла ПС;

– удостоверять достигнутое качество и надежность функционирования ПС в процессе их испытаний и **сертификации** перед передачей в регулярную эксплуатацию;

– оперативно выявлять последствия дефектов программ и данных и восстанавливать нормальное, надежное функционирование комплексов программ.

Комплексное, скоординированное применение этих методов и средств в процессе создания, развития и применения ПС позволяет исключать некоторые виды угроз или значительно ослаблять их влияние. Тем самым уровень достигаемой надежности ПС становится предсказуемым и управляемым, непосредственно зависящим от ресурсов, выделяемых на его достижение, а главное от качества и эффективности технологии, используемой на всех этапах жизненного цикла ПС.

Итак, основными методами предотвращения угроз надежности являются: предотвращение ошибок проектирования за счет использования CАSE-технологий, систематическое тестирование и обязательная сертификация. Оперативными методами повышения надежности являются: временная, программная и информационная избыточность.

### **Систематизация принципов и методов обеспечения надежности в соответствии с их целью**

Все принципы и методы обеспечения надежности в соответствии с их целью можно разбить на четыре группы: предупреждение ошибок, обнаружение ошибок, исправление ошибок и обеспечение устойчивости к ошибкам. К первой группе относятся принципы и методы, позволяющие минимизировать или вообще исключить ошибки. Методы второй группы сосредоточивают внимание на функциях самого программного обеспечения, помогающих выявлять ошибки. К третьей группе относятся функции программного обеспечения, предназначенные для исправления ошибок или их последствий. Устойчивость к ошибкам (четвертая группа) – это мера способности системы программного обеспечения продолжать функционирование при наличии ошибок.

**Предупреждение ошибок.** К этой группе относятся принципы и методы, цель которых – не допустить появления ошибок в готовой программе. Большинство методов концентрируется на отдельных процессах перевода и направлено на предупреждение ошибок в этих процессах. Их можно разбить на следующие категории:

1 методы, позволяющие справиться со сложностью, свести ее к минимуму, так как это – главная причина ошибок перевода;

2 методы достижения большей точности при переводе;

3 методы улучшения обмена информацией;

4 методы немедленного обнаружения и устранения ошибок. Эти методы направлены на обнаружение ошибок на каждом шаге перевода, не откладывая до тестирования программы после ее написания.

Очевидно, что предупреждение ошибок – оптимальный путь к достижению надежности программного обеспечения.

Лучший способ обеспечить надежность – прежде всего не допустить возникновения ошибок. Гарантировать отсутствие ошибок, однако, невозможно никогда. Другие три группы методов опираются на предположение, что ошибки все-таки будут.

**Обнаружение ошибок.** Если предполагать, что в программном обеспечении какие–то ошибки все же будут, то лучшая (после предупреждения ошибок) стратегия – включить средства обнаружения ошибок в само программное обеспечение.

Большинство методов направлено по возможности на незамедлительное обнаружение сбоев. Немедленное обнаружение имеет два преимущества: можно минимизировать влияние ошибки и последующие затруднения для человека, которому придется извлекать информацию о ней, находить ее и исправлять.

Меры по обнаружению ошибок можно разбить на две подгруппы: пассивные попытки обнаружить симптомы ошибки в процессе «обычной» работы программного обеспечения и активные попытки программной системы периодически обследовать свое состояние в поисках признаков ошибок.

Пассивное обнаружение. Меры по обнаружению ошибок могут быть приняты на нескольких структурных уровнях программной системы. Здесь мы будем рассматривать уровень подсистем, или компонентов, т.е. нас будут интересовать меры по обнаружению симптомов ошибок, предпринимаемые при переходе от одного компонента к другому, а также внутри компонента. Все это, конечно, применимо также к отдельным модулям внутри компонента.

Разрабатывая эти меры, мы будем опираться на следующее.

1 Взаимное недоверие. Каждый из компонентов должен предполагать, что все другие содержат ошибки. Когда он получает какие-нибудь данные от другого компонента или из источника вне системы, он должен предполагать, что данные могут быть неправильными, и пытаться найти в них ошибки.

2 Немедленное обнаружение. Ошибки необходимо обнаружить как можно раньше. Это не только ограничивает наносимый ими ущерб, но и значительно упрощает задачу отладки.

3 Избыточность. Все средства обнаружения ошибок основаны на некоторой форме избыточности (явной или неявной).

Когда разрабатываются меры по обнаружению ошибок, важно принять согласованную стратегию для всей системы. Действия, предпринимаемые после обнаружения ошибки в программном обеспечении, должны быть единообразными для всех компонентов системы. Это ставит вопрос о том, какие именно действия следует предпринять, когда ошибка обнаружена. Наилучшее решение – немедленно завершить выполнение программы или (в случае операционной системы) перевести центральный процессор в состояние ожидания. С точки зрения предоставления человеку, отлаживающему программу, например системному программисту, самых благоприятных условий для диагностики ошибок немедленное завершение представляется наилучшей стратегией. Конечно, во многих системах подобная стратегия бывает нецелесообразной (например, может оказаться, что приостанавливать работу системы нельзя). В таком случае используется метод регистрации ошибок. Описание симптомов ошибки и «моментальный снимок» состояния системы сохраняются во внешнем файле, после чего система может продолжать работу. Этот файл позднее будет изучен обслуживающим персоналом.

Всегда, когда это возможно, лучше приостановить выполнение программы, чем регистрировать ошибки (либо обеспечить как дополнительную возможность работу системы в любом из этих режимов). Различие между этими методами проиллюстрируем на способах выявления причин возникающего иногда скрежета вашего автомобиля. Если автомеханик находится на заднем сиденье, то он может обследовать состояние машины в тот момент, когда скрежет возникает. Если вы выбираете метод регистрации ошибок, задача диагностики станет сложнее.

Активное обнаружение ошибок. Не все ошибки можно выявить пассивными методами, поскольку эти методы обнаруживают ошибку лишь тогда, когда ее симптомы подвергаются соответствующей проверке. Можно делать и дополнительные проверки, если спроектировать специальные программные средства для активного поиска признаков ошибок в системе. Такие средства называются средствами активного обнаружения ошибок.

Активные средства обнаружения ошибок обычно объединяются в диагностический монитор: параллельный процесс, который периодически анализирует состояние системы с целью обнаружить ошибку. Большие программные системы, управляющие ресурсами, часто содержат ошибки, приводящие к потере ресурсов на длительное время. Например, управление памятью операционной системы сдает блоки памяти «в аренду» программам пользователей и другим частям операционной системы. Ошибка в этих самых «других частях» системы может иногда вести к неправильной работе блока управления памятью, занимающегося возвратом сданной ранее в аренду памяти, что вызывает медленное вырождение системы.

Диагностический монитор можно реализовать как периодически выполняемую задачу (например, она планируется на каждый час) либо как задачу с низким приоритетом, которая планируется для выполнения в то время, когда система переходит в состояние ожидания. Как и прежде, выполняемые монитором конкретные проверки зависят от специфики системы, но некоторые идеи будут понятны из примеров. Монитор может обследовать основную память, чтобы обнаружить блоки памяти, не выделенные ни одной из выполняемых задач и не включенные в системный список свободной памяти. Он может проверять также необычные ситуации: например, процесс не планировался для выполнения в течение некоторого разумного интервала времени. Монитор может осуществлять поиск «затерявшихся» внутри системы сообщений или операций ввода–вывода, которые необычно долгое время остаются незавершенными, участков памяти на диске, которые не помечены как выделенные и не включены в список свободной памяти, а также различного рода странностей в файлах данных.

Иногда желательно, чтобы в чрезвычайных обстоятельствах монитор выполнял диагностические тесты системы. Он может вызывать определенные системные функции, сравнивая их результат с заранее определенным и проверяя, насколько разумно время выполнения. Монитор может также периодически предъявлять системе «пустые» или «легкие» задания, чтобы убедиться, что система функционирует хотя бы самым примитивным образом.

**Исправление ошибок.** Следующий шаг – методы исправления ошибок; после того как ошибка обнаружена, либо она сама, либо ее последствия должны быть исправлены программным обеспечением. Исправление ошибок самой системой – плодотворный метод проектирования надежных систем аппаратного обеспечения. Некоторые устройства способны обнаружить неисправные компоненты и перейти к использованию идентичных запасных. Аналогичные методы неприменимы к программному обеспечению вследствие глубоких внутренних различий между сбоями аппаратуры и ошибками в программах. Если некоторый программный модуль содержит ошибку, идентичные «запасные» модули также будут содержать ту же ошибку.

Другой подход к исправлению связан с попытками восстановить разрушения, вызванные ошибками, например искажения записей в базе данных или управляющих таблицах системы. Польза от методов борьбы с искажениями ограничена, поскольку предполагается, что разработчик заранее предугадает несколько возможных типов искажений и предусмотрит программно реализуемые функции для их устранения. Это похоже на парадокс, поскольку, если знать заранее, какие ошибки возникнут, можно было бы принять дополнительные меры по их предупреждению. Если методы ликвидации последствий сбоев не могут быть обобщены для работы со многими типами искажений, лучше всего направлять силы и средства на предупреждение ошибок. Вместо того, чтобы, разрабатывая операционную систему, оснащать ее средствами обнаружения и восстановления цепочки искаженных таблиц или управляющих блоков, следовало бы лучше спроектировать систему так, чтобы только один модуль имел доступ к этой цепочке, а затем настойчиво пытаться убедиться в правильности этого модуля.

**Устойчивость к ошибкам.** Методы этой группы ставят своей целью обеспечить функционирование программной системы при наличии в ней ошибок. Они разбиваются на три подгруппы: динамическая избыточность, методы отступления и методы изоляции ошибок.

1 Истоки концепции динамической избыточности лежат в проектировании аппаратного обеспечения. Один из подходов к динамической избыточности – метод голосования. Данные обрабатываются независимо несколькими идентичными устройствами, и результаты сравниваются. Если большинство устройств выработало одинаковый результат, этот результат и считается правильным. И опять, вследствие особой природы ошибок в программном обеспечении ошибка, имеющаяся в копии программного модуля, будет также присутствовать во всех других его копиях, поэтому идея голосования здесь, видимо, неприемлема. Предлагаемый иногда подход к решению этой проблемы состоит в том, чтобы иметь несколько неидентичных копий модуля. Это значит, что все копии выполняют одну и ту же функцию, но либо реализуют различные алгоритмы, либо созданы разными разработчиками. Этот подход бесперспективен по следующим причинам. Часто трудно получить существенно разные версии модуля, выполняющие одинаковые функции. Кроме того, возникает необходимость в дополнительном программном обеспечении для организации выполнения этих версий параллельно или последовательно и сравнения результатов. Это дополнительное программное обеспечение повышает уровень сложности системы, что, конечно, противоречит основной идее предупреждения ошибок – стремиться в первую очередь минимизировать сложность.

Второй подход к динамической избыточности – выполнять эти запасные копии только тогда, когда результаты, полученные с помощью основной копии, признаны неправильными. Если это происходит, система автоматически вызывает запасную копию. Если и ее результаты неправильны, вызывается другая запасная копия и т. д.

2. Вторая подгруппа методов обеспечения устойчивости к ошибкам называется методами отступления или сокращенного обслуживания. Эти методы приемлемы обычно лишь тогда, когда для системы программного обеспечения существенно важно корректно закончить работу. Например, если ошибка оказывается в системе, управляющей технологическими процессами, и в результате эта система выходит из строя, то может быть загружен и выполнен особый фрагмент программы, призванный подстраховать систему и обеспечить безаварийное завершение всех управляемых системой процессов. Аналогичные средства часто необходимы в операционных системах. Если операционная система обнаруживает, что вот-вот выйдет из строя, она может загрузить аварийный фрагмент, ответственный за оповещение пользователей у терминалов о предстоящем сбое и за сохранение всех критических для системы данных.

3. Последняя подгруппа – методы изоляции ошибок. Основная их идея – не дать как можно большей части последствиям ошибки выйти за пределы системы программного обеспечения: так чтобы, если ошибка возникнет, то не вся система оказалась неработоспособной; отключаются лишь отдельные функции в системе либо некоторые ее пользователи. Например, во многих операционных системах изолируются ошибки отдельных пользователей, так что сбой влияет лишь на некоторое подмножество пользователей, а система в целом продолжает функционировать. В телефонных переключательных системах для восстановления после ошибки, чтобы не рисковать выходом из строя всей системы, просто разрывают телефонную связь. Другие методы изоляции ошибок связаны с защитой каждой из программ в системе от ошибок других программ. Ошибка в прикладной программе, выполняемой под управлением операционной системы, должна оказывать влияние только на эту программу. Она не должна сказываться на операционной системе или других программах, функционирующих в этой системе.

В большой вычислительной системе изоляция программ является ключевым фактором, гарантирующим, что отказы в программе одного пользователя не приведут к отказам в программах других пользователей или к полному выводу системы из строя. Основные правила изоляции ошибок перечислены ниже. Хотя в формулировке многих из них употребляются слова «операционная система», они применимы к любой программе (будь то операционная система, монитор телеобработки или подсистема управления файлами), которая занята обслуживанием других программ.

1 Прикладная программа не должна иметь возможности непосредственно ссылаться на другую прикладную программу или данные в другой программе и изменять их.

2 Прикладная программа не должна иметь возможности непосредственно ссылаться на программы или данные операционной системы и изменять их. Связь между двумя программами (или программой и операционной системой) может быть разрешена только при условии использования четко определенных сопряжений и только в случае, когда обе программы дают согласие на эту связь.

3 Прикладные программы и их данные должны быть защищены от операционной системы до такой степени, чтобы ошибки в операционной системе не могли привести к случайному изменению прикладных программ или их данных.

4 Операционная система должна защищать все прикладные программы и данные от случайного их изменения операторами системы или обслуживающим персоналом.

5 Прикладные программы не должны иметь возможности ни остановить систему, ни вынудить ее изменить другую прикладную программу или ее данные. Когда прикладная программа обращается к операционной системе, должна проверяться допустимость всех параметров. Прикладная программа не должна иметь возможности изменить эти параметры между моментами проверки и реального их использования операционной системой.

6 Никакие системные данные, непосредственно доступные прикладным программам, не должны влиять на функционирование операционной системы. Ошибка в прикладной программе, вследствие которой содержимое этой памяти может быть случайно изменено, приводит в конце концов к сбою системы.

7 Прикладные программы не должны иметь возможности в обход операционной системы прямо использовать управляемые ею аппаратные ресурсы. Прикладные программы не должны прямо вызывать компоненты операционной системы, предназначенные для использования только ее подсистемами.

8 Компоненты операционной системы должны быть изолированы друг от друга так, чтобы ошибка в одной из них не привела к изменению других компонентов или их данных.

9 Если операционная система обнаруживает ошибку в себе самой, она должна попытаться ограничить влияние этой ошибки одной прикладной программой и в крайнем случае прекратить выполнение только этой программы.

10 Операционная система должна давать прикладным программам возможность по требованию исправлять обнаруженные в них ошибки, а не безоговорочно прекращать их выполнение.

Реализация многих из этих принципов влияет на архитектуру лежащего в основе системы аппаратного обеспечения.

Из рассмотренных выше трех подгрупп методов обеспечения устойчивости к ошибкам только третья, изоляция ошибок, применима для большинства систем программного обеспечения.

Важное обстоятельство, касающееся всех четырех подходов, состоит в том, что обнаружение, исправление ошибок и устойчивость к ошибкам в некотором отношении противоположны методам предупреждения ошибок. В частности, обнаружение, исправление и устойчивость требуют дополнительных функций от самого программного обеспечения. Тем самым не только увеличивается сложность готовой системы, но и появляется возможность внести новые ошибки при реализации этих функций. Как правило, все рассматриваемые методы предупреждения и многие методы обнаружения ошибок применимы к любому программному проекту. Методы исправления ошибок и обеспечения устойчивости применяются не очень широко. Это, однако, зависит от области приложения. Если рассматривается, скажем, система реального времени, то ясно, что она должна сохранить работоспособность и при наличии ошибок, а тогда могут оказаться желательными и методы исправления и обеспечения устойчивости. К системам такого типа относятся телефонные переключательные системы, системы управления технологическими процессами, аэрокосмические и авиационные диспетчерские системы и операционные системы широкого назначения.